SNEAK: Faster Interactive Search-based SE

Andre Lustosa, Jaydeep Patel, Venkata Sai Teja Malapati, Tim Menzies, *Fellow, IEEE*

Abstract—When AI tools can generate many solutions, some human preference must be applied to determine which solution is relevant to the current project. One way to find those preferences is *interactive search-based software engineering* (iSBSE) where humans can influence the search process.

This paper argues that when optimizing a model using human-in-the-loop, data mining methods such as our SNEAK tool (that recurses into divisions of the data) perform better than standard iSBSE methods (that mutates multiple candidate solutions over many generations). For our case studies, SNEAK runs faster, asks fewer questions, achieves better solutions (that are within 3% of the best solutions seen in our sample space), and scales to large problems (in our experiments, models with 1000 variables can be explored with half a dozen interactions where, each time, we ask only four questions).

Accordingly, we recommend SNEAK as a baseline against which future iSBSE work should be compared. To facilitate that, all our scripts are online at https://github.com/ai-se/sneak.

Index Terms—Interactive Search-based Software Engineering, Optimization, Model Reasoning, Cognitive Load Reduction

✦

1 INTRODUCTION

When models are too complex, AI tools can be of great assistance. For example, the SCRUM model offered by Mendonca et al. [1] has 128 project management options and nearly 300 constraints (e.g., if sprints last two weeks, then each individual task must take less than 10 days of programming). Those constraints are so complicated that less than 2% of the 2^{128} possible choices are acceptable to that model. When reasoning about this large space of constrained options, AI tools can be very useful. For example, the PicoSAT [2] SAT solver can find tens of millions of satisfying solutions to the SCRUM model.

But now there is a new problem: too many solutions. When PicoSAT finds millions of solutions, human preferences must be applied to find which solutions are more acceptable for the current project. One way to find those preferences is *interactive search-based software engineering)* (iSBSE) where humans can influence the search process [3]. iSBSE is an active area of research (see examples in Table 2) with iSBSE methods typically evolving 100+ candidate solutions (or more) over dozens of generations. In that process, it can be overwhelming for humans to be asked too many complex questions about all those mutants of all those solutions. Worse, as discussed below, those methods have issues with scalability and effectiveness.

An alternate approach, as implemented in our SNEAK tool, is to replace that evolutionary approach with a data mining method that knows how to find minimal models. To reduce that cognitive fatigue, our SNEAK algorithm, takes as input a list of candidate solutions extracted from (e.g.) SAT Solvers or simulations. Next, SNEAK's recursive

- A. Lustosa and T. Menzies are with the Department of Computer Science, North Carolina State University, Raleigh, USA.
  E-mail:alustos@ncsu.edu, timm@ieee.org
- Jaydeep Patel and Venkata Sai Teja Malapati are Masters students at the Department of Computer Science, North Carolina State University, Raleigh, USA.
  E-mail:jpatel33@ncsu.edu, vmalapa@ncsu.edu

bi-clustering algorithm prunes half the candidates at each level of recursion via asking users for their preference (where some entropy-based feature selection culls away uninteresting questions). Finally SNEAK uses a state-of-the-art multi-objective optimizer [4] to select a final solution from the surviving few.

To assess our novel approach, this paper compares SNEAK to a state-of-the-art iSBSE method [6]. For a range of different SE models (SCRUM, online BILLING, the POM3 agile modeling, the XOMO waterfall model, and several artificially generated models), SNEAK asks fewer questions while achieving better results. For example, even after evaluating 10,000s of examples, the prior state-of-the art could only ever find solutions 12 times worse than SNEAK for the SCRUM model (and in that case, SNEAK found those solution after asking just 25 questions).

The rest of this paper is structured according to guidelines from Wohlin & Runeson et al. [7] on how to structure experiments in software engineering. According to that advice, we have presented our case studies in §4.1 defined our experiments in §5, where we detail it's design and execution followed by our analysis in §6. Our conclusion is SNEAK should be considered as a baseline against which future iSBSE work should be compared. Also we suggest to the community that it could be insightful to further explore the central hypothesis of this work:

> **The SNEAK hypothesis**: *When optimizing a model using human-in-the-loop, data mining methods (that recurses into divisions of the data) do better than evolutionary methods (that evolve multiple candidate solutions over many generations)*

Note that the specialized terms used in this paper are presented in Table 1.

2 BACKGROUND

2.1 Search-based SE

In this paper, a model describing a space of options (e.g. for configuring a SCRUM project) is searched for

Features	The configurable variables of any model in this study (described in §4.1).
Interaction	An interaction here can always be read as some query sent to the human for data collection.
Loop	All of the steps that are run between interactions with SNEAK (described in §3) are a loop.
Questions	The two sets of features presented to the user at each iteration.
D2H	After sorting all solutions according to Zitzler's predicate [5] (described in §3.2.7) we calculate the percentage distance from each solution to the best.
Human Preferences	New preferences are recorded whenever humans answer questions.
Human Preference as a Goal	One of the optimization goals of this algorithm is to produce a solution containing the largest possible subset from *Human Preferences*.
Oracle Effort	Time required by an oracle to answer the questions asked by the iSBSE system .
Computational Effort	CPU-Time, or time taken by an algorithm top process information.
Standard Methods	By "standard methods", we mean evolutionary SBSE algorithms.

TABLE 1: Frequently used terminology

TABLE 2: 24 recent iSBSE papers (for more details, see the literature review of Table 3).

1 Requirements	: [6, 8, 9, 10, 11]
2 Analysis and design	: [12, 13, 14, 15, 16, 17, 18, 19]
3 Code implementation	: [20]
4 Testing and verification	: [21, 22, 23]
5 Distribution and maintenance	: [24, 25, 26, 27, 28, 29, 30]
6 Prescriptive process model	: [6, 8, 9]
7 Other.	: [10, 11, 12, 13, 14, 15, 16, 17, 18]
	: [19, 20, 23, 24, 25, 26, 27, 29, 30]

valid choices. One complexity is that in any engineering discipline, including software engineering, it is common to trade-off between multiple competing goals. For example:

- For requirements engineers, we can find the *least* cost mitigations that enable *most* requirements [31].
- For project managers, we can explore software process models to find options that deliver *more* code in *less* time with *fewer bugs* [32].
- For developers, we might tune data miners looking for ways to find *more* bugs in *fewer* lines of code (thereby reducing the human inspection effort required once the learner has finished [33]).

Search-based SE (SBSE) is an automatic method for exploring that trade-space [34]. Unlike traditional optimization methods (e.g. SIMPLEX [35]), search-based SE makes no assumption that all goals are achievable. Instead, search-based SE makes a more realistic assumption that to win on some goals might mean trading off on other goals.

Recent papers in the SBSE arena have explored many issues including *project management* with tools for software effort estimation [36, 37, 38, 39, 40, 41, 42, 43, 44, 45], and managing human resources [46]; *requirements optimization* [31, 47]; *software design* with tools for software architecture optimization [48] and extraction of products from very large product lines [45]; *software security* and intrusion detection [49, 50]; *software quality* with tools for software detect prediction [51, 52, 53, 54, 55, 56, 57] and test case generation [58]; *software configuration* [59, 60, 46]; *text mining* with tools for reasoning about StackOverflow [61, 62]; topic modeling [62, 63]; defect reports [62]; and software artifact search [64].

There are many ways to implement SBSE and this paper adopts the "oversampling" methods base on the SWAY system from Chen et al. [4]. SWAY compared two approaches to optimization:

- A *traditional approach* that mutated (say) 10^2 individuals across (e.g.) 10^2 generations;
- The SWAY *oversampling approach* that builds generation$_0$ with 10,000, individuals, which are then recursively clustered and pruned (by removing sub-trees whose roots are dominated by a near neighbor).

In studies with optimizing decisions within product lines, Chen et al. were able to show that SWAY's oversampling approach found solutions as good, or better, than the traditional approaches. This is a natural approach for SNEAK since Chen et al. also based their empirical work on software project lines. Also, as mentioned in the introduction, our problems start with SAT solvers, such as PicoSAT, or specific model generators which over-generate a large number of candidate solutions, or generative methods provided by specific models.

While an interesting prototype, SWAY suffers from some serious drawbacks. Both SWAY and SNEAK recursively partition the data, but SWAY's greedy search always commits to the top-most partition (without checking if any other partition is better) whereas SNEAK takes a global approach. Specifically, after all the data is clustered, SNEAK reflects over all the nodes to find which sub-clusters "best" split the data (and "best" means "splits the data such that some attribute range most separates the two splits").

Apart from greedy-vs-global, a more significant issue is that SWAY does not know how to talk to people. When dividing its data, SWAY needs some oracle to decide if some example A is better than example B. At that point, the best SWAY can do is present *all* attributes of each example to a human. Further, even if it took advice from humans, for highly constrained models, that advice might be wrong. Consider the SCRUM model used in this paper, with its 128 binary options and almost 300 constraints. As mentioned above, only a few percent of the possible solutions are valid. This means that even if a human oracle can find meaning in some combination of 128 attributes, then they still might make a decision that is incorrect. Clearly, before we can unleash SWAY on humans, we need a method for constraining what it asks, while at the same time ensuring that all the collected answers come from the space of valid solutions. Hence, the motivation for this paper.

2.2 Interactive Search-based SE

One issue with standard SBSE is all its conclusions are "black-box"; i.e. these algorithms run and produce results, even if users have no understanding or input into how

		Technique (see Table 4)	P-Type (see Table 2)	Year	Venue	Citations
Interactive requirements prioritization using a genetic algorithm	[10]	3	1	2013	TSE	107
Interactive, evolutionary search in upstream object-oriented class design	[16]	3	2	2010	TSE	102
Recommendation system for software refactoring using innovization and interactive dynamic ...	[25]	3	5	2014	ASE	79
Putting the developer in-the-loop: an interactive GA for software re-modularization	[30]	3	5	2012	SSBSE	74
Elegant object-oriented software design via interactive, evolutionary computation	[15]	3	2	2012	SMC	61
Interactive genetic algorithms for user interface design	[18]	3	2	2007	CEC	55
Imagine: a tool for generating HTML style sheets with an interactive genetic algorithm ...	[19]	3	2	1999	SMC	53
Improving feature location practice with multi-faceted interactive exploration	[29]	3	5	2013	ICSE	51
Model refactoring using interactive genetic algorithm	[28]	3	7	2016	SSBSE	49
Interactive and guided architectural refactoring with search-based recommendation	[24]	2	5	2016	SIGSOFT	46
On the use of machine learning and search-based software engineering for ill-defined fitness...	[27]	3	5	2014	SSBSE	46
Interactive ant colony optimization (iACO) for early lifecycle software design	[12]	4	2	2014	Springer	33
An Architecture based on interactive optimization and machine learning applied to the next ...	[6]	3	1 & 6	2017	ASE	32
Using an SMT Solver for Interactive Requirements Prioritization	[11]	1	1	2011	ACM	25
An initial industrial evaluation of interactive search-based testing for embedded software	[22]	3	4	2015	ASC	20
Incorporating user preferences in ant colony optimization for the next release problem	[8]	4	1 & 6	2016	ASE	16
Tester interactivity makes a difference in search-based software testing: A controlled experiment	[21]	3	4	2016	ISBST	14
OPTI-SELECT: An interactive tool for user-in-the-loop feature selection in software product lines	[13]	3	2	2014	SPLC	13
Interactive software release planning with preferences base	[9]	3	1 & 6	2015	SSBSE	11
Objective re-weighting to guide an interactive search based software testing system	[23]	3	4	2013	ICMLA	11
An empirical investigation of search-based computational support for conceptual software ...	[17]	3	2	2009	SMC	9
A validation study regarding a generative approach in choosing appropriate colors for .	[12]	3	2	2015	Springer	4
Interactive code smells detection: An initial investigation	[26]	3	5	2016	SSBSE	3
This paper ⟹		1	2 & 6	2022	TBA	

TABLE 3: Overview of iSBSE research. Technique is indexed based on the IDs of Table 4 . P-Type (Problem type) is indexed based on the IDs of Table 2

TABLE 4: Note: Search Techniques Seen Used in iSBSE.

ID	Single Objective	Multi-objective	Many-objective	Other	Example system (discussed in §2.2)	
1	**Exact**	4.1%	0%	0%	0%	[11]
	Metaheuristic					
2	*Single-solution based*	4.1%	0%	0%	0%	[24]
3	*Evolutionary computation*	61.5%	20.5%	4.1%	4.1%	[6]
4	*Swarm intelligence*	8.3%	0%	0%	0%	[8]

those results are obtained. Interactive SBSE is a variant of SBSE that tries to include humans in the reasoning process. More specifically, given a range of options, iSBSE tries to find which preferences are most important to users.

One question we are frequently asked is "should AI ignore human preferences lest those preferences force optimizers into some sub-optimal region of the total space?". In reply, we point out that this has not been the experience of the iSBSE field. Instead, the usual result is that competitive optimization results can be achieved while also respecting user preferences. Certainly, there are limits to adopting all user preferences[1]. But one of the more interesting results shown below is that we can respect user preferences within our product lines, while still achieving optimization results that are within 1% to 3% of the best in our sample.

A little mathematics offers one explanation for our results. In the experiments below, using the SCRUM model, our algorithms select 20 preferences within a space of 128 boolean variables. That is, the space acceptable to our oracles is $2^{20}/2^{128} \approx$ a trillionth of the decision space. iSBSE can respect human preferences while still achieving good optimization results since even with all those preferences set, there are still (*1 - 1/trillion*) of the options available to the optimizer.

This fraction $2^{20}/2^{128}$ also explains why iSBSE should be seen as an essential part of human+AI decision making:

- iSBSE is essential to human-AI interaction since it stops human preferences from being ignored.
- To see this, recall the fraction $2^{20}/2^{128}$ of important preferences within our SCRUM model. When important user preferences fall within one-trillionth of the total space then, without iSBSE, it is vanishingly unlikely that a fully automated algorithm will select those preferred options.

To highlight these mathematics on the effectiveness of standard SBSE algorithms, we baselined our approach to a standard non-interactive Genetic Algorithm when appropriate.

For all these reasons, we explore the current iSBSE research seen in Table 3. To build that table, we started with a prominent paper in that area: specifically, the TSE'19 paper by Ramirez et al. [3]. Then following standard practices as established by Skoglund and Runeson [65] we explored all the papers referenced by that paper or all subsequent papers that referenced it, via the forward snowballing method [66]. This set was pruned to those that proposed methods for interactive search in search-based software engineering. This yielded the 24 papers of Table 3. Some details on those 24 papers are shown in Table 2 and Table 4.

2.3 Open Issues with iSBSE

Many current iSBSE methods have issues with:

- Cognitive fatigue;
- Scalability;
- and Effectiveness.

1. For example, at sea-level on the planet Earth, no user preference can change gravity from being 9.8 ms^2.

iSBSE methods can lead to cognitive fatigue when they overwhelm humans with too many, overly elaborate questions. Standard SBSE methods generate and review thousands to millions of options via a process of mutation, crossover, and selection (repeated for dozens to hundreds of generations). Humans cannot quickly nor accurately review that much material. Valerdi [67] reports that his panels of human experts required three meetings (three hours each) to reach convergence on the influence of 10 variables on 10 examples (in the domain of cost estimation). Hence, according to Valerdi, it would be unreasonable to expect humans to quickly assess (e.g.) 128 options in a SCRUM model.

Even if humans felt they could comment on (e.g.) 128 options, can their assessment be trusted? Shackleford et al. [68] warn that humans' cognitive fatigue leads to errors in human decisions about what variables are most influential. Takagi [69, 70, 71] notes that human cognitive fatigue can be decreased by:

- Reducing I the number of *interactions* (where, at each interaction, we ask the user questions)
- Reducing S the *size* of each interaction (number of questions asked per interaction)

Many of the iSBSE methods in Table 3 implement Takagai's advice. In summary, given some logs of the decisions made so far, iSBSE infers what possibilities can be ignored. As shown by the left-hand-side column of Table 3, there are many ways to do this. For example, Palma et al. use a state-of-the-art constraint solver (the MAX-SMT algorithm as a sub-routine inside iSBSE). We call this approach "exact" since it explores all solutions to a requirement model expressed as a set of constraints via extensive user interaction ($I = 25$ to 100). In their method, they elicit pair-wise comparisons of partial model solutions to decrease their "uncertainty" value about the final proposed solution. The decisions made by humans in evaluating these pairs of characteristics adjust elements of the optimizer's problem formulation towards a better solution. While an interesting approach, we note that Palma et al. seem nervous about the scalability of their methods. The biggest model they ever report processing has 50 variables and not that many constraints. While, shown below, we report results within a 1000 variable model showing that it scales relative to a small 9 variable model.

As to other iSBSE methods from Table 3, Lin et al. [24] presented a "single solution based" method for code refactoring tasks. Their method Refactoring Navigator (RN) calculates and recommends refactoring "*paths*" from the starting point to the target, each *path* being a sequence of atomic refactorings. The user examines the recommended steps and can accept, reject, or ignore them interactively. These interactions will then be used as feedback to calculate the next recommendation. In this case, the concept of S represents the size of each refactoring,and how many different refactorings a user has to evaluate at each interaction. Like Palma et al., Lin et al. are cautious about the scalability of their methods.

Araújo et al. [6] use the evolutionary computation architecture of Fig. 1. This architecture combined an interactive genetic algorithm with a machine learner.

Initially, humans are utilized to evaluate examples. But once there are enough examples to train a learner, Araújo et al. evaluate candidates via the learner's recommendations. Their approach starts with a user setup of the optimization goal (e.g. how many generations it should run for and the number of human interactions the user wishes to perform). This number needs to be large enough to allow for the learning model to converge.

This process of Figure 1 continues until the number of human selected interactions is reached, after which the Learning Model is trained and replaces the human. Their algorithm shows the user the entire candidate solution for evaluation, which is then given a numerical answer. This answer is then included in the formula for calculating the fitness of an individual.

Ferreira et al. [8] proposed a "Swarm intelligence architecture" based on Araújo et al. [6] where the user inputs their preferences on each of the features for a particular model. The algorithm then changes its search approach on the feature space and provides the user with a candidate solution. On each interaction, the user can either accept the solution, terminating the algorithm or they can reset their preferences for each of the model's features to query for a different solution.

Another issue with current iSBSE methods is that the existing techniques have issues with effectiveness. Sometimes, the heuristics used by the above methods result in fewer questions to the user, but also worse solutions. For example:

- Ferreira et al. report an average decrease of 13% in their 50 feature model experiments' maximum performance score, where their technique was only able to reach 13% of the best solutions generated by their methods.
- When reasoning about a model with many constraints, most of the computational time will be spent generating and evaluating solutions that might not hold to all of the constraints. Hsu et al. [72] said one main cause of human fatigue are the occasions where the user's preferred result does not exist in the search space. Thus, subjecting humans to possible evaluations on invalid solutions can also contribute to fatigue.
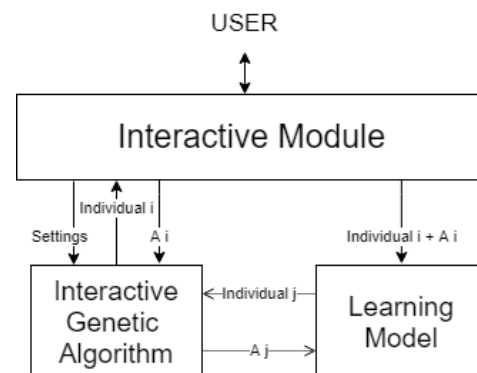

Fig. 1: The Araújo et al. architecture [6]

With one exception, other iSBSE methods are specific to the syntactic form of specific models– which makes it hard to express our new models in the form of that older, prior work. For example, the input to the method of Palma et al. [11] required a table that ranked the importance of each feature. In principle, we could have generated such a table from our models (e.g. do some random sampling, then sort each input variable according to the output score). However, that ran the risk of introducing a threat to validity (i.e. that we had translated our models incorrectly). For example, consider the random sampling procedure defined in the last sentence. That procedure has numerous control parameters (e.g. number of samples, how to rank instances when multiple goals are being perused; etc). The wrong settings to those parameters would change the nature of the inputs and the validity of the solution. Therefore, for our experiments, we prefer the generality of Araújo et al.'s [6] approach.

Apart from generality, there are two other reasons to focus on the methods of Araújo et al.:

- As shown in Table 3, Araújo et al. is part of the *evolutionary computation* group, which represents the majority of prior work. Hence, by comparing SNEAK with Araújo et al., we are comparing to the standard iSBSE methods seen in the literature.
- For future work, Araújo et al. recommended exploring the software product lines used in this study. They say "[Our] conceptual architecture can be considered sufficiently generic to be adopted in other software engineering scenarios tackled by SBSE, such as *feature selection in software product lines...*" [6].

Accordingly, we baseline SNEAK against Araújo et al.

2.4 Software Product Lines

To evaluate SNEAK, we require some prior state-of-the-art tool (e.g. Araújo et al.) as well as a set of models to explore. For this case study, we will use two constraint-heavy software product lines (SPL) describing (a) SCRUM process options within an agile software projects; (b) an online BILLING system; and six instances of two other configuration problems, which contain fewer constraints.

Apart from these eight real world examples, in order to test for the generality of these methods and their behavior on larger models, we have used the Model Generation tool

Fig. 2: A product line for a simple phone. This example continues in Fig. 3.

from the SPLOT-Research web site[2]. With this, we have generated eight more models. (Described in §4.1).

Rocha and Fantinato [73] comment that product lines are a generalization of the SPL architecture design for business process management. Kang's feature models [74] are often used to represent a product line. That representation combines in a tree of features and constraints defining the multiple legal variations of a product [75]. For example, Fig. 2 shows such a tree of options for a simple cell phone. If expressed in CNF (conjunctive normal form), that same model can be expressed as per Fig. 3a.

Tools like the PicoSAT solver take CNF formula expressed in the Dimacs format of Fig. 3b, and output variable settings that satisfy the model constraints. A standard strategy (used by PicoSAT) is that, after one solution is found, its negation is added back to the model and the SAT Solver is rerun to find a different solution. As a very simple model, the phone model contains only 14 possible solutions and they are instantly found by PicoSAT in Fig. 3c. Each solution is a valid product extracted from a product line like Fig. 2.

Fig. 5 shows Mendonca et al.'s model of options within SCRUM projects. This model was written using information from Schwaber et al.'s SCRUM Guide [76]. SCRUM is defined as a lightweight framework that helps organizations to deal with complex problems through adaptive solutions [76]. The model is comprised of options detailing the structure of the development team as well as their roles in the project. It also includes the configuration of various SCRUM activities and artifacts. The model contains many mandatory features, focusing the selections further down the leaves. The cross-tree constraints create a very complex environment that rules out many possibilities. Specifically: less than 2% of the 2^{128} possible projects satisfy those 256 constraints. Hence, humans need tool support to help them navigate that confusing space of options.

3 THE SNEAK ALGORITHM (DETAILS)

Algorithm 3 shows many of the internal details of SNEAK. SNEAK takes a binary dataset comprising of multiple valid solutions to a given problem as input and outputs (a) a binary tree of clustered solutions; from which it reports (b) the best "solution". Each "solution" is a list of true, false assignments to each feature (and if a feature is "true" then the feature is active/present on the final product).

An interesting detail of SNEAK is that it defers multi-objective optimization until *after* it prunes away solutions that violate some of the learned user preferences. For that final optimization step, SNEAK uses Chen et al.'s SWAY algorithm [4] (described in §2.1).

The rest of this section offers details on the inner workings of SNEAK. For our discussion, we adopt the following convention: any superscript number in round brackets (e.g. [(1)]) refers to a line number in Algorithm 1.

3.1 Clustering Binary Variables

Initially, we tried a simple Euclidean distance measure. This performed poorly, especially as the number of features in
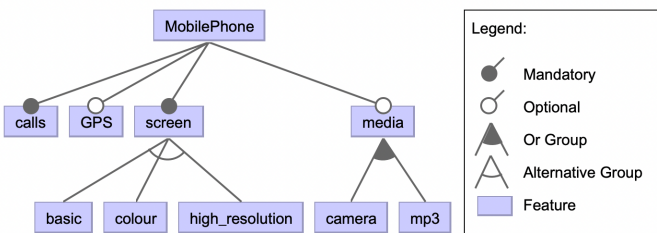
$(MobilePhone)$
$\wedge(screen \vee \neg basic)$
$\wedge(screen \vee \neg colour)$
$\wedge(screen \vee \neg high\_resolution)$
$\wedge(basic \vee colour \vee high\_resolution \vee \neg screen)$
$\wedge(\neg basic \vee \neg colour)$
$\wedge(\neg basic \vee \neg high\_resolution)$
$\wedge(\neg colour \vee \neg high\_resolution)$
$\wedge(media \vee \neg camera)$
$\wedge(media \vee \neg mp3)$
$\wedge(camera \vee mp3 \vee \neg media)$
$\wedge(MobilePhone \vee \neg calls)$
$\wedge(MobilePhone \vee \neg GPS)$
$\wedge(MobilePhone \vee \neg screen)$
$\wedge(MobilePhone \vee \neg media)$
$\wedge(calls \vee \neg MobilePhone)$
$\wedge(screen \vee \neg MobilePhone)$
$\wedge(\neg GPS \vee \neg basic)$
$\wedge(high\_resolution \vee \neg camera)$

(a) CNF formula for Fig. 2.

```
1 : c 1 mp3
2 : c 2 colour
3 : c 3 high_resolution
4 : c 4 calls
5 : c 5 screen
6 : c 6 GPS
7 : c 7 media
8 : c 8 basic
9 : c 9 camera
10 : c 10 MobilePhone
11 : p cnf 10 19
12 : 10 0.        # i.e. MobilePhone=true
13 : 5 -8 0       # i.e.. screen=true or basic=false
14 : 5 -2 0
15 : 5 -3 0
16 : 8 2 3 -5 0
17 : -8 -2 0
18 : -8 -3 0
19 : -2 -3 0
20 : 7 -9 0
21 : 7 -1 0
22 : 9 1 -7 0
23 : 10 -4 0
24 : 10 -6 0
25 : 10 -5 0
26 : 10 -7 0
27 : 4 -10 0
28 : 5 -10 0
29 : -6 -8 0
30 : 3 -9 0
```

(b) The Dimacs representation of Fig. 3a.

- calls screen basic
- high_resolution calls screen
- high_resolution calls screen media camera
- high_resolution calls screen GPS media camera
- high_resolution calls screen GPS
- colour calls screen GPS
- colour calls screen
- mp3 colour calls screen GPS media
- mp3 colour calls screen media
- mp3 calls screen media basic
- mp3 high_resolution calls screen media
- mp3 high_resolution calls screen media camera
- mp3 high_resolution calls screen GPS media
- mp3 high_resolution calls screen GPS media camera

(c) PicoSAT inputs Fig. 3b to output multiple solutions. Each line shown here is one valid product within Fig. 2.

Fig. 3: In Figure3.b, lines 1 to 11 are variable names. Line 11 says there are 19 clauses with 10 variables. After that, a "0" denotes end of a clause; each clause is a disjunction; negatives denote negation; all clauses form one conjunction.

Algorithm 1 SNEAK

   Input Items, Model
   Output Solution
   Parameter Settings
   Requires Select, Pruning, Oracle, Optimizer
1: $tree \leftarrow RBI(Items, Settings)$ ▷ Defined in Algorithm. 2
2: **foreach** $feature \in Model$ **do**
3:    $feature.good \leftarrow Good(feature)$ ▷ Defined in Eq. 3
4: **foreach** $node \in tree$ **do**
5:    $node.good \leftarrow Good(node)$ ▷ Defined in Eq. 4
6: **while** any $good$ **do**
7:    $target \leftarrow sort(nodes)[-1]$
8:    $tree \leftarrow Pruning(target, tree, Settings.best)$
9:    **foreach** $feature \in Model$ **do**
10:      $feature.good \leftarrow Good(feature)$
11:    **foreach** $node \in tree$ **do**
12:      $node.good \leftarrow Good(node)$
13: $survivors \leftarrow tree.elements$
14: $Solution \leftarrow Optimizer(survivors, Model, Settings.Budget)$
15: **return** $Solution$

Algorithm 2 Recursive Bi-Clustering (RBI)

   Input Items
   Output TreeCluster
   Parameter Settings
1: **if** $|Items| < Settings.Enough$ **then**
2:    **return** $MakeTree(Items)$ ▷ MakeTree uses a standard binary tree node
3: $W, E, W\_I, E\_I \leftarrow SPLIT(Items, Settings.Bins)$ ▷ See Alg. 3
4: $EastNode \leftarrow RBI(E\_I), Settings$
5: $WestNode \leftarrow RBI(W\_I), Settings$
6: $TreeCluster \leftarrow MakeTree(E, W, EastNode, WestNode)$
7: **return** $TreeCluster$

Algorithm 3 SPLIT for binary decision spaces. From [4]

   Input Items
   Output [west, east], [C1, C2]
   Parameter Bins = 10

1: $pivot \leftarrow$ any random item   ▷ 'zero' point for radial distances
2: $max\_r \leftarrow -\infty$
3: $minx\_r \leftarrow \infty$
4: **foreach** $x \in Items$ **do**   ▷ collect info for radial distance
5:    $x.r \leftarrow \sum_{n=1}^{||x||} x\_n$
6:    $x.d \leftarrow \sum_{n=1}^{||x||} x\_n - pivot$ ▷ distance from radial center
7:    **if** $x.r > max\_r$ **then return** $max\_r = x.r$
8:    **if** $x.r < min\_r$ **then return** $min\_r = x.r$
9: **foreach** $x \in Items$ **do**
10:    $x.r = \frac{x.r - min\_r}{max\_r - min\_r + 10^{-32}}$
11: $R \leftarrow \{i.r | i \in Items\}$
12: **foreach** $k \in R$ **do**
13:    $g = \{i | i.r = k\}$
14:    sort g by $x.d$. $g \leftarrow x_1, x_2, ..., x_{|g|}$
15:    **foreach** $i \in [1, |g|]$ **do** $x_i.\theta \leftarrow \frac{2\pi i}{|g|}$ ▷ angular angle from pivot
16: $thk \leftarrow \frac{max\_r}{Bins}$
17: $west \leftarrow []$
18: $east \leftarrow []$
19: $C1 \leftarrow []$
20: $C2 \leftarrow []$
21: **foreach** $a \leq Bins$ **do**   ▷ divide data in two
22:    $g = \{i | (a-1)thk \leq i.r \leq (a)thk\}$
23:    sort g by $g.\theta$
24:    $east \leftarrow MIN\_g$
25:    $west \leftarrow MAX\_g$
26:    $C1 \leftarrow \{i | i \in g \ \& \ i.\theta \leq \pi\}$
27:    $C2 \leftarrow \{i | i \in g \ \& \ i.\theta > \pi\}$
   return $[west, east], [C1, C2]$

Setting	Default	Notes
$sample$	= 10,000	Initial number of generated solutions
$enough$	= $sample^{0.5}$	Recursion stops when subtrees have less than $enough$ items.
$pruning$	= function	Decide which nodes be used for the next question, collect that answer, delete anything that contradicts that answer.
$best$	= 6	Maximum number of questions to ask at each point of the loop.
$stop$	= !$good$	The algorithm terminates when there are no longer nodes or features that we can ask questions on.
$oracle$	= function	The oracle decides which features are preferred.
$select$	= function	When recursion terminates, the Zitzler's predicate [5] is used as the "Better" function on Chen et al.'s SWAY [4] to return the preferred solution.
$budget$	= standard	When the interactive part of SNEAK terminates, budget controls how we select the best solution within those the survivors.
t	= 20	Number of random trials (used in evaluation).

TABLE 5: Control options for SNEAK.

the a model increased. All our data is non-numeric, or at the very least transformed in such a way to become binary. In each solution, a feature is either selected or not, which we initially (and naively) modeled as with true=1 and false=0. This was incorrect since there is no meaning to fractional values (e.g.) 0.5.

For boolean variables, SNEAK uses Chen et al.'s boolean radial distance metric [4]: In this approach, solutions are recursively biclustered as follows. One solution is selected (at random) to be the "pivot" and all other solutions are divided in two (where the division point is the median distance of all points to the pivot). The algorithm then recurses on each division. In the Chen et al. approach, distance is calculated as follows.

- All solutions get assigned a "radius" equal to the count of how many "trues" appear in the solution.
- Within each radius, each solution gets assigned an angle θ according to its overlap with the pivot (see line 15 of Alg. 3).

Once the algorithm completes we recurse on each half until we have leaves with less than *enough* items. This tree is then reviewed using the methods described below in §3.2.4.

3.2 Control Parameters

Table 5 lists the control parameters for SNEAK. This section explains each of those parameters.

3.2.1 Samples

Samples measures how many valid candidate solutions have been generated through various means, either using a SAT solver, or another solution generator for non CNF models, before the use of SNEAK. Once generated, SNEAK structures that space using a recursive biclustering algorithm (i.e. at each level of recursion, divide the data into two groups through Alg. 3, then recurse on each group through Alg. 2). This generates a tree, where each node contains multiple valid solutions.

3.2.2 Enough

The recursive biclustering repeats until leaf clusters have less than *enough* solutions as seen in Alg. 2.

3.2.3 Pruning

The *pruning* function [8] inputs the tree (generated above) and outputs a smaller tree that contains solutions that do not contradict user preferences. During this function, the oracle defined below is asked questions. In order to minimize the number of times we ask the oracle questions, we use the tools of §3.2.4.

What the oracle does not know is that answer is being used by SNEAK to evaluate and prune the two children of a given node in the tree. The oracle's answers are used to:

- Prune *all* the solutions in one of the sub-trees;
- Prune *some* of the solutions in the other surviving sub-tree (those that contradict the answers),

Pruning is applied repeatedly until the *stop* criteria of §3.2.5, is satisfied.

Once pruning terminates, the surviving solutions are searched for those that maximize (or minimize) the domain goals. For more on that process, see §3.2.7.

3.2.4 Best

Best controls how many questions (at most) are asked at each loop of the algorithm. [8] SNEAK ranks all the features and nodes of the remaining tree at some point in the loop using four heuristics:

- Refraction;
- Entropy;
- Depth of first difference;
- Differences.

If more than one feature has a maximum rank, then SNEAK selects up to *best* maximum scoring features, at random. In practice:

- At the start of the interactive step, SNEAK asks the maximum number of questions
- Deeper down into the interactions, that number decreases down to one.

Refraction just says "never ask the same thing twice". Each feature f_i is assigned a value $asked[f_i] = 0$ (initially) and then $asked[f_i] = 1$ if we ask the user about whether or not to use that feature.

The same is done about individual nodes in the tree. With each of them being assigned the same value $asked[n_i] = 0$ (initially) and then $asked[n_i] = 1$ if we query the user on that particular node.

Entropy is a measure of the variability of a symbolic distribution [77]. At each point of the loop, the subset of N remaining candidate solutions at that point mention features $F = f_1, f_2, ...$ which, in turn, have settings *true* or *false* at frequencies $n, N - n$ respectively. The entropy of each feature is a measure of the effort required to recreate that distribution; i.e.

$$e[f_i] = -\left(\frac{n}{N}\right)\log_2\left(\frac{n}{N}\right) - \left(\frac{N-n}{N}\right)\log_2\left(\frac{N-n}{N}\right) \quad (1)$$

The feature with maximum entropy e is the one which, if we decide to use, will exclude the largest number of remaining solutions.

Depth of first difference is a measure of how early in our tree-like structure a certain feature differs in the sub-trees in the subset of N remaining candidate solutions. At each point of the loop, the depth of each feature is a normalized count of the first difference of that given feature on the tree i.e.

$$d[f_i] = \frac{depth[f_i]}{|cur\_tree|} \quad (2)$$

The feature with minimum depth e is the one which, if we decide to use, will exclude the largest number of remaining sub-trees.

Differences: SNEAK ranks all the features in order to select which features will be asked at that point in time through the following equation[10]:

$$good_{f_i} = e[f_i] \star (1 - d[f_i]) \star (1 - asked[f_i]) \quad (3)$$

In this equation, $e[f_i]$ and $d[f_i]$ come from Equation 1 and Equation 2 respectively. Let $\Delta[f_i] = 1$ if some feature f_i has different settings in the representatives of each side a given surviving node (and zero otherwise). To select a the

most relevant node to ask at each level of the loop, SNEAK selects the node that maximizes[13]:

$$good_{n_i} = \frac{\sum_i^{|f|}(\Delta[f_i] \star good_{f_i}) \star (1 - asked[n_i])}{\sum_i^{|f|} \Delta[f_i]} \quad (4)$$

In this equation, $good_{f_i}$ comes from Equation 3. $good$ evaluates to zero if a feature has been *asked* before, or in the case of $good_{n_i}$, if there is no difference in its variables between the two sub-trees (i.e. $\Delta[f_i] = 0$ for all variables).

3.2.5 Stop

The loop repeats until the algorithm runs out of interesting nodes or features to ask, i.e. both *good* methods return 0 for the entire remaining set of candidate solutions [6].

3.2.6 Oracle

Some of our processing is stochastic so we must repeat that inference, say, 20 times (to sample the behavior of our algorithm). For that kind of test, we used an artificial oracle since, otherwise, we would be exhausting our human subject matter experts. The artificial oracle was implemented as follows. If the oracle is asked about feature f_i, then it will pick true or false at random, then caches that decision (for the remainder of that run).

On top of that, we also used human oracles to assess our models. For those subjects, we use a team of 20 developers from a large Brazilian I.T. organization (see §6.3).

3.2.7 Select

When all the above terminates, the initial tree of solutions is now much smaller (since many of the decisions that contradict user preferences have been removed). For single-objective optimization problem, a simple sorting function can rank goals between candidate solutions. However, when dealing with multi-objective reasoning, those candidates must be ranked across many goals.

Binary domination will say that one sample is better than the other if at least one goal is better. But when dealing with three or more goals, binary domination will have trouble distinguishing samples [78].

Sayyad et al. [79] have recommended that the Zitzler's *continuous domination predicate* [5] should be used in such cases. Therefore, this paper utilizes Zitzler's predicate, as described in Algorithm 4, to rank the candidate solutions in order to evaluate the effectiveness of SNEAK.

This predicate is then used as the evaluation function required by SWAY [4] to do our final multi-objective

---

Algorithm 4 Zitzler's predicate for sorting multi goals

> **Input** A, B (Two Solutions)
> **Output** A ≤ B (Binary)
> **Parameter** W (Vector of weights for each goal)
> **Requires** Eval (Model objective-evaluation function)

1: $A.evals \leftarrow Eval(A)$ ▷ Evaluate solution A
2: $B.evals \leftarrow Eval(B)$ ▷ Evaluate solution B
3: $S_a = 0, S_b = 0, n = len(A.evals)$
4: **foreach** $index, V_a, V_b \in enumerate(A, B)$ **do**   ▷ loop through all goals
5:     $S_a = S_a - EXP(w[index] * (V_a - V_b)/n)$ ▷ EXP(x) = e^x
6: $S_b = S_b - EXP(w[index] * (V_b - V_a)/n)$
7: **return** $S_a/n \leq S_b/n$

---

optimization. This further reduces the space from $\sqrt{samples}$ to $\sqrt{\sqrt{samples}}$. After, we evaluate the surviving solutions (after SWAY) and rank them according to the Zitzler's predicate. From there, we can find the best one. These best solutions are then scored via the *distance to heaven* (*d2h*) measure. To find *d2h*, we assume an omniscient oracle that can evaluate all examples. Note that this omniscient oracle, that evaluates all the solutions, is only needed when assessing SNEAK's output. Otherwise, SNEAK only evaluates a small fraction of the examples.

Using this oracle, we sort every example ever seen during the entire inference from best to worse, using Algorithm 4. Given a vector Z containing all solutions sorted from best to worst (via Zitzler), the d2h of a specific solution s placed in index i is:

$$d2h_s = i/|Z| \quad (5)$$

Note that this number ranges from 0 to 1 and *lower* numbers are *better* (since they are closest to the best).

3.2.8 Budget

The budget is a parameter that controls how many examples we ask the oracle to evaluate. The current default is to use our **standard** strategy described in §3.2.7. This strategy that requires our oracle to evaluate $O(\sqrt{N})$ of the remaining examples (since we evaluate all the surviving examples after SWAY has been applied). It future work, it would be insightful to explore other budgeting policies.

4 METHODS

4.1 Case Studies: Models Used in this Study

Table 6 offers statistics on all the models used in this study. For details on these models, see the rest of this section.

In our results, the following observation will become significant. The models are *OSP2, POM3A, POM3B, POM3C, FLIGHT* are small and have no constraints (so all solutions generated to these models are valid). All the other models are much larger have many constraints. Hence, as shown in Fig. 4, only 1% to 3% the solutions generated for these models are valid.

4.1.1 XOMO

XOMO, introduced by Menzies et al. [80] combines four different COCOMO-like software process models in order to calculate *project risk*, *development effort*, *predicted defects* and *development time*. XOMO's optimization goal is then to minimize all these metrics. This is a non-trivial task since the objectives are obviously conflicting (e.g.: to reduce project risk and the number of defects, you should raise development effort and development time). The three XOMO models studied here represent three NASA systems: OSP2 (the orbital space plane, version2), GROUND (ground systems) and FLIGHT (flight systems). Of all the models studied here, XOMO is the simplest since it only requires the optimization of a dozen attributes, or less.

	"Free" variables (i.e. values that can be adjusted).	Constraint ratio	Number of goals
OSP2	6	0	5
POM3A	9	0	4
POM3B	9	0	4
POM3C	9	0	4
FLIGHT	11	0	5
GROUND	12	0	5
BILLING	88	1.02	4
125FEAT	125	0.25	4
SCRUM	128	0.97	4
250FEAT	250	0.25	4
.25 C.D.	500	0.25	4
.50 C.D.	500	0.50	4
.75 C.D.	500	0.75	4
1.00 C.D.	500	1.00	4
500FEAT	500	0.25	4
1000FEAT	1000	0.25	4

TABLE 6: Number of variables being optimized and constraint ratio. For the top models, the constraint ratio is zero (since every solution from those models is valid). For the other models, this ratio is the number of constraints per clauses.
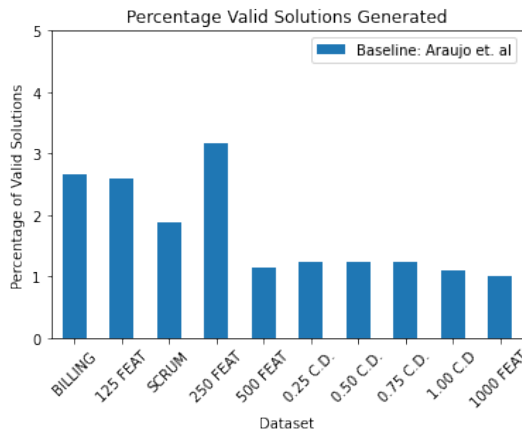
Fig. 4: Valid Solutions over 20 runs (Technical aside: to handle these invalid solutions, we added a post-processor to the Araújo et al. algorithm that rejected invalid solutions.)

4.1.2 POM3

POM3 is a model for exploring the management process of agile development [81, 82, 83]. The objective of POM3 is to find an effective configuration for nine continuous variables in order to:

- Increase completion rates (delivery more features, sooner);
- Reduce the time spent by one team, waiting for the products of another;
- And decrease the overall cost.

According to Chen et al. [4], POM is far more complex than XOMO (longer to run, much harder to simultaneously optimize all its goals). Also, here we run three variants of POM explored by Chen et al (POM3A, POM3B, POM3C). These variants describe different projects and, measured in term of complexity, Chen et al. report that POM3A is the least complex and POM3C is the most complex of them all.

4.1.3 Software Product Lines

In this paper, we have used models obtained from the SXFM SPLOT-Research web site[3]. In these models, we seek to minimize costs, efforts, and predicted defects while maximizing features associated with previous success.

For example, the SCRUM model of Fig. 5 which is an SPL defining the possible legal configurations of the SCRUM organization framework. As shown on this figure, the SCRUM model contains 128 features and more than 250 constraints. These constraints are so complicated that out of the 2^{128} possible solutions less than 2% is valid.

Another model from the SPLOT web site is BILLING. This is a simpler model that defines valid configurations for a billing software system. Containing 88 features and 166 constraints, BILLING is not as large as SCRUM. That said, less than 2% of its 2^{88} configurations satisfy its internal constraints.

SPLOT also contains tools for artificially generating models. This feature is useful for stress testing an algorithm by (say) generating models of increasing complexity. We built eight such models:

- In 125FEAT, 250FEAT, 500FEAT and 1000FEAT, the number of features was *increased* while the ratio of constraints to features was kept *constant*.
- In 0.25 C.D, 0.75 C.D, 0.50 C.D and 1.00 C.D, the number of features was kept *constant* (at 500) while the ratio of constraints to features was *increased*.

These ten models were taken from the SXFM format and converted into the Dimacs format using the FeatureIDE [84]. The Dimacs format is a standard interface to most SAT solvers, through that we have used PicoSAT to extract a database of valid solutions to each.

4.2 Algorithms

SNEAK is implemented in Python 3.8. That code is available on-line [4]. That site contains CSV files listing tens of thousands of solutions to all our models. Those solutions were generated either via PicoSAT v0.6.3 which can be installed via "pip3 install pycosat==0.6.3". Or through the generative methods from the POM3 and XOMO models.

Acuri and Briand [85] recommend that algorithms need to be compared against some simple baseline. For that purpose, we use a Non Interactive Genetic Algorithm (NGA). This is a simple one-point cross-over genetic algorithm that mutates 1% of the attributes using initial population of 100 valid solutions, and is ran for 100 generations (generating 100 new samples each time). The code for that GA is available at our Github site [5].

For our work with the Araújo et al. system, they did not offer a reproduction package for their work. Hence, we reimplemented their code based on their description [6].

Another under-specified part of the Araújo et al.'s work was how many evaluations they used within their evolutionary algorithms. To say the least, this complicates our ability to compare this work with their results. Hence,

3. http://www.splot-research.org/
4. https://github.com/ai-se/sneak
5. https://github.com/ai-se/sneak/blob/main/NGA.ipynb
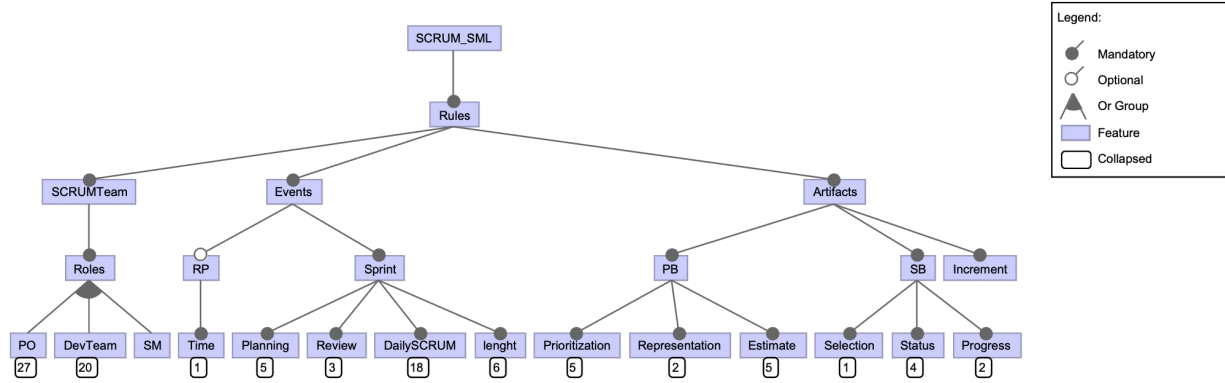6. https://github.com/ai-se/sneak/blob/main/BASELINE.ipynb

Fig. 5: The SCRUM Feature Model. Summary of the top level of the SCRUM feature model. The entire model, in XML format, is 166 lines long (see https://github.com/ai-se/sneak/blob/main/SXFM/Scrum.xml). When translated into conjunctive normal form, all the disjunctions combine together to generate a large set of CNF clauses with hundreds of mutually exclusive feature settings. In all, the CNF version of this model has 128 options with 256 constraints.

to be as fair as possible to prior work, we will give Araújo et al. (and NGA) a large evaluation budget (the 100 individuals mutated for 100 generations, as recommended by Golderg [86]).

5 DESIGN AND EXECUTION

In order to evaluate how effective is this approach for iSBSE we must measure:

- The number of interactions (i.e. how many times we go to a user to ask questions);
- The size of each interaction (i.e. do we show partial or full solutions to users at each interaction);
- How effective is our search process (measured by the Zitzler predicate in Equation 5).
- How much do humans trust and accept the selected final solution;
- What is the oracle cost of the approach (i.e. scalability both in computational and effort of the oracle)

When using an automatic oracle, all these experiments are repeated 20 times with different random number seeds. For the record, the median runtimes for one repeat (in our largest model with 1000 nodes) were 2 minutes with SNEAK and 162 minutes using Araújo et al. or NGA. As mentioned in the last section, the evalaution budget was under-specified in the rig of Araújo et al. Hence, we take care *not* to make conclusions based on runtimes between the algorithms.

What is more relevant for iSBSE applications is the *oracle cost* of running these algorithms. Therefore, we take care below to report the number of interactions *as well as* the size of each interaction between the algorithm and the oracle.

6 ANALYSIS

6.1 Research Questions

Our results explore three research questions:

- **RQ1**: How effective is SNEAK at finding "good" solutions?
- **RQ2**: How human acceptable are SNEAK's solutions?
- **RQ3**: Does SNEAK reduce the oracle cost of iSBSE?

6.2 RQ1: How effective is SNEAK at finding "good" solutions?

Fig. 6a shows that when Araújo et al. evaluates its 10,000 candidates, it pauses around 50 times to ask users some questions. As shown in Fig. 6b, these questions cover all the attributes in the model. On the other hand, SNEAK pauses around 20 times, and when it does, it asks far fewer questions than Araújo et al.

Fig. 7 comments on the effectiveness of the questions asked by SNEAK. Usually, SNEAK can find results within 2 to 3% of the best solution ever seen for a model. More specifically, in Fig. 7:

- For all models, the simple GA usually performs worse that everything else.
- For Araújo et al., its median scores are better than SNEAK for the unconstrained models (on the left-hand-side). But once we move to models with constraints (i.e. from GROUND, then too the left), SNEAK out-performs Araújo et al. Sometimes, those wins are very large indeed: e.g. see BILLING here SNEAK's median *d2h* scores are an order of magnitude better than the other algorithms.
- As to SNEAK, when it loses to Araújo et al., the difference is usually very small (to be precise, the actual *d2h* performance deltas from the best results are $\{1, 2, 2, 3, 3, 3\}\%$).

Hence our answer to RQ1 is "for large and complex models with many constraints, SNEAK is the clear winner" (of the systems studied here). To be sure, sometimes SNEAK is defeated by other methods– but only by a very small amount and only in simple models (that are both very small and have no constraints).

More importantly, measured in terms of the human cost to find a solution, SNEAK is the clear winner. We say that that cost is *number of interactions* times *number of questions per interaction*. Using Fig. 7, we can compare the cost for our different ssytems. SNEAK's oracle cost was 4% of Araújo et al. for some models (the unconstrained ones) and less than 1% for others (the constrained ones).
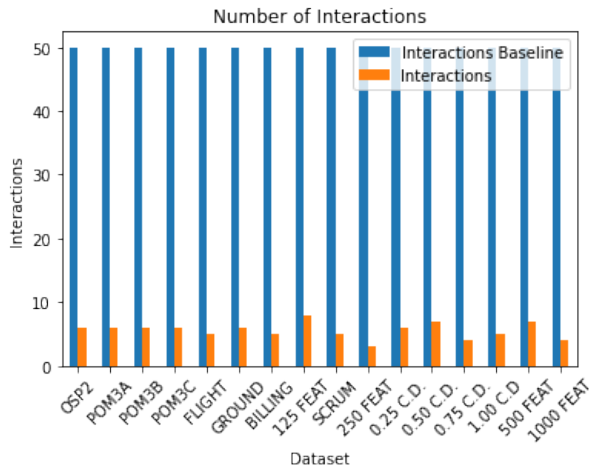
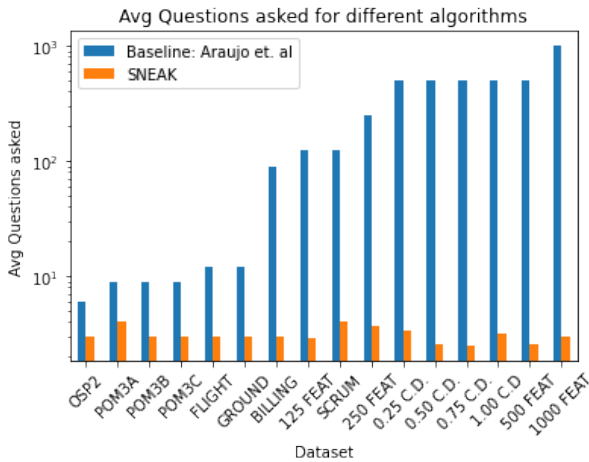**Fig. 6a**: *I = Number of interactions (medians from 20 runs)*

Fig. 6b:*S = Median questions (in 20 repeats) asked per interaction (Log scale on Y axis)*

Fig. 6: RQ1 results. Any iSBSE tool will, I times, ask the user about S features. Shown here are the I, S numbers seen in 20 runs (different random number seeds each time).
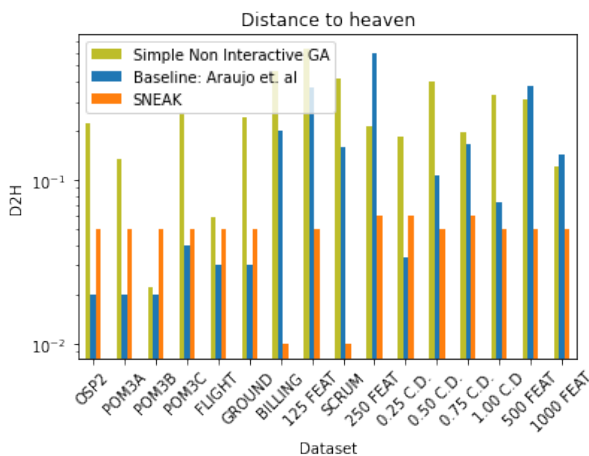

Fig. 7: RQ1 results. Median *d2h* values seen over 20 runs (Log scale on Y axis). For a definition of *d2h*, see Equation 5 in §3.2.7. Note that *lower* values are *better*.

6.3 RQ2: How well accepted are SNEAK's solutions to experts?

As mentioned in §2.2, users may have preferences for only a tiny fraction of the total space. Hence, it is possible that automated tools like SNEAK will roam far from what a user condones as acceptable. RQ2 checks for that problem.

The following experiment was performed in accordance to North Carolina State University's Institutional Review Board (protocol #24233). We reached out to our contacts in the Brazilian I.T. community where one manager was kind enough to grant access to 20 of her developers, on the condition we took no more than one hour of their time (for initial briefing and running the experiments).

Our choice of subjects lead to certain decisions for our experimental design. For example, the experiment could not include any execution of the Araújo et al. method (since that was too slow). Also, we had to use a model with attributes that our subjects understood. In consultation with the manager, we reviewed several models (before speaking to subjects) and the decision there was that POM was the most approachable for our subjects.

For this experiment, subjects were selected by their manager. All subjects had at least 4 years of experience in the field and 3 years of experience in being in an agile team. Subjects were made aware that the manager endorsed their participation in this study. No added incentives were offered to subjects except a commitment that if in future they wanted to use the tool, then we would make it freely available and support them for that use. Subjects were guaranteed anonymity and that their manager would not see their specific results (to ensure that, the experiment did not collect logins or names or IP addresses).

Prior to collecting data, we conducted a short (less than 20 minutes) virtual meeting with all 20 participants (which included the manager of that group, who also worked as a subject). Subjects were introduced to the goals of the work and were briefed on the model being used. After that, subjects had 48 hours to complete the experiment and, for that period, they were asked not to talk to each other about their experiences. While performing the experiment, the subjects were observed by one of the authors via Microsoft Teams. These observers did not speak the entire time of the experiment. Subjects installed and ran our software locally on their own machines. During the experiment subjects where shown two configurations (see Fig. 8) and asked to rate each one (using a score 0 to 5, where 0 was worst).

During the experiment, we employed deception. Subjects were told that all solutions came from SNEAK. But actually, one recommendations was generated via picking at random across a space of 10,000 valid solutions. These solutions were presented in a randomized order so that the SNEAK solutions did not always appear in the same place on the screen.

As we can see in Fig. 9 in most cases the experts preferred SNEAK's solutions. The median score for SNEAK's solutions was 4 while the median score for randomly selected solutions was 2. Hence for RQ2, we answer "yes, for models we could show to our subjects in their available time, SNEAK's solutions are acceptable".
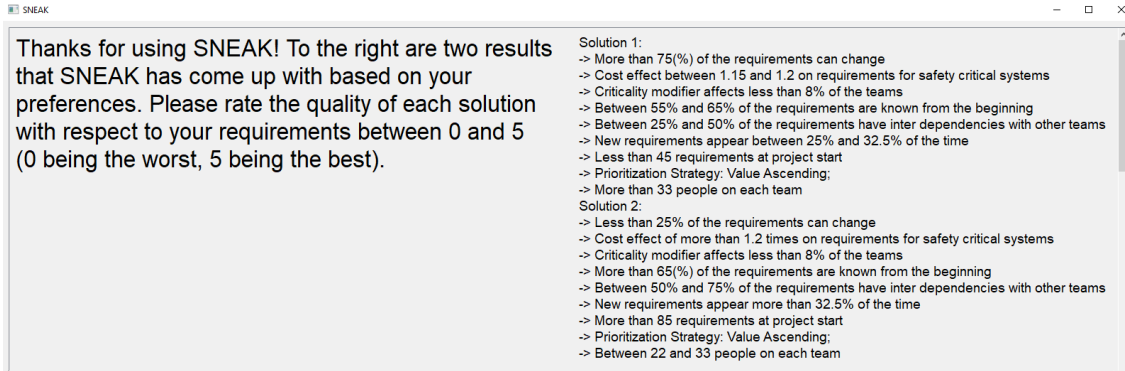
Fig. 8: RQ2 results: One of the screens seen during the human oracle experiments.
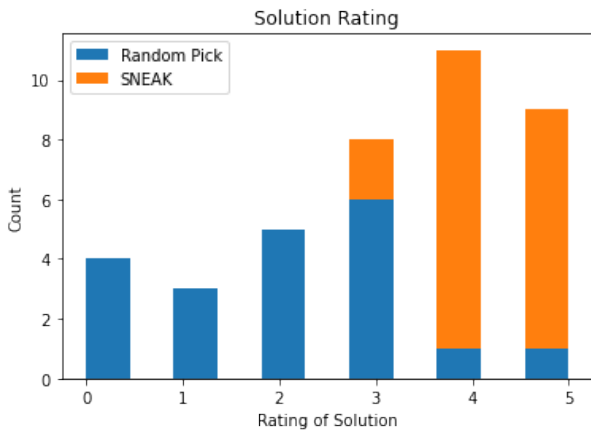

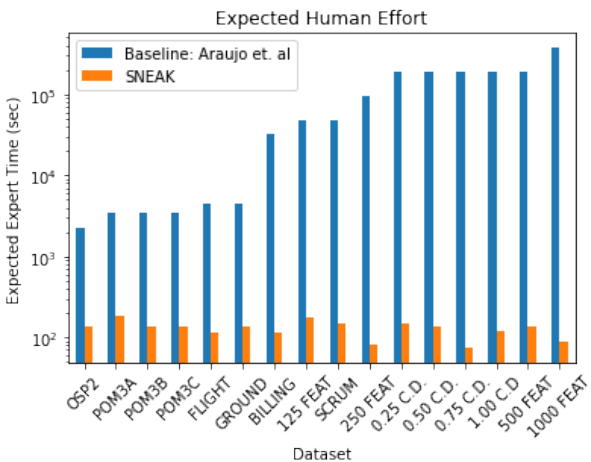**Fig. 9: RQ2 results. Distribution of ratings given by human experts to presented solutions.**

Fig. 10: RQ3 results: Expectation of human oracle effort to explore each model (calculated using the assumption of 7.5s per variable evaluation, log scale on Y axis)

6.4 RQ3: Does SNEAK reduce the oracle cost of iSBSE?

During the RQ2 data collection, we found uniformity across our 20 users. Specifically, we observed that for the POM3A model, our subjects averaged 7.5 seconds per variable evaluation. Using that information, we can offer some estimations on the time required for humans to interact with our models using SNEAK or the Araújo et al. method.

Fig. 10 shows the calculated expected human oracle effort with the assumption of 7.5 seconds to answer questions about a variable. Note that SNEAK requires orders of magnitude less time to process models than our comparison iSBSE system:

- From POM3, SNEAK is 18 times faster;
- For SCRUM, SNEAK is 350 times faster;
- For our largest model with 1000 nodes, SNEAK is 4200 times faster.

6.5 Other Issues

Apart from our research questions, we have other reasons to recommend SNEAK over the methods of Araújo et al.: 100% of all the solutions explored by SNEAK are valid. The reason for this is simple: we let PicoSAT, or the model generative tool, generate valid solutions, then we down-sample from that space.

This is a significant and important feature of SNEAK. Other iSBSE methods can generate far few valid solutions when dealing with models containing many constraints. Fig. 4 showed what happens when we take the solutions generated by Araújo et al.'s genetic algorithm [6] for the SPL models, then applied the model constraints to those solutions. As seen in that figure, most of the solutions generated by their methods are not valid (since their genetic algorithm does not take into consideration those constraints to build the new solution).

6.6 Threats to Validity

As with any empirical study, biases can affect the final results. Therefore, any conclusions made from this work must be considered with the following issues in mind:

Evaluation Bias - In our human experiments, we had the participation of 20 software engineers using SNEAK to select for a good POM3A [81, 82, 83] configuration (a model for exploring the management process of agile development). Although the number of professionals could be higher to provide us a more reliable evaluation of the approach, we have mitigated this through selecting only professionals with at least 4 years of experience in the field and 3 years of experience in being in an agile team.

Another concern is the fact that given the time these professionals conceded to us towards the experiments we had expected them not to be able to complete a run with the baseline algorithm. To mitigate this and provide a fair comparison we used a similar process as the one described by Araújo et al. [6] to evaluate our final solutions. Through this we were able to infer the human oracle effort required by the baseline algorithm as described in **RQ4**.

Sampling Bias - This threatens any empirical study using datasets. i.e, what works here may not work everywhere. For example, we have applied SNEAK to two real-life models of different software product lines, six XOMO and POM3 models and eight artificially generated feature models of varying characteristics (see §4.1). However, the behavior of SNEAK to very much larger models (i.e. hundreds of thousands of features) still needs to be evaluated.

Another concern is that if the models studied here are "trivial" in some sense, then there may be very little value added by SNEAK. We do not believe these these models are trivial for several reasons. Firstly, these models have a very large internal search space. In fact, for all practical purposes, that search space is so large that it cannot be enumerated. Consider, for example the SCRUM model with its 128 binary options. Assuming that the cross-tree constraints wipe out tens of millions of possible solutions, that still leaves a space at least 2^{120} solutions. Our experience with PicoSAT is that takes 12 hours to generate 100 million solutions[7]. Hence 2^{120} solutions would require $\approx 2 * 10^{25}$ years to enumerate all solutions.

Secondly, as discussed in RQ3, measured in terms of human evaluation time, these models are exceedingly non-trivial. Fig. 10 reports the time required by humans to evaluate these models. When assisted by tools *other* than SNEAK, humans need all day (8 hours+) to review all the candidates generated by those standard methods.

Parameter bias - SNEAK is controlled by numerous hyperparameters (see Table 5). While this work shows that the selected parameters on Table 5 are capable of generating solutions as effective than prior state-of-the-art with a much reduced computational and human effort, we believe that future work on the parameter variation may prove to give better results. That said, in defense of our current settings, we note that we can find solutions within **1% to 3%** of the best seen in our sample.

Algorithm bias - To the best of our knowledge the selected baseline from Araújo et al. [6] was the only available iSBSE tool generic enough to be used as a comparison against SNEAK. An adaptation of SNEAK to more niche problems in the iSBSE field may prove to be fruitful from a comparison standpoint.

7 DISCUSSION: WHY DOES SNEAK WORK?

This section presents a mathematical analysis that suggests that SNEAK's results are not-so-unexpected. This analysis makes several limiting assumptions, which means that its conclusions are an optimistic, best case scenario. Those assumptions are:

- All solutions can be accessed by the optimizer
- The evaluation predicate offers a linear sort over those solutions (i.e. in that sort, item i is better than item $i + 1$);
- We are only concerned with the *rank* of what is found and not the *absolute value* of the distance to the best solution.

Given a linear sort l of all possible choices, where l_1 is best and l_i is worse than l_{i+1}, we can compute the confidence c in finding a choice in the top p percent of that sort after n random samples. Since an event p will not happen after n trials is $(1 - p)^n$, then our confidence in seeing it after n trials is:

$$c = 1 - (1 - p)^n$$

This expression can be re-arranged to find the number of samples required to find a choice:

$$n(c, p) = \frac{log(1 - c)}{log(1 - p)}$$

To further reduce n, we note that SNEAK does a binary chop through its data. Hence, in a manner similar to Hoare's quickselect algorithm[8] we need only explore $2 * \log_2 n$ of the above n samples.

(Technical note: We evaluate the pair that fall on the boundary points of our divisions, and not all points. Hence we say $2 * \log_2 n$ and not quickselect's usual case of n evaluations [87].)

From Fig. 7, we see that SNEAK's median distance to heaven (to the top-ranked solution) was 1% to 3%. Therefore, by the above equations, we need less than 20 random samples (with a binary chop) to achieve our results with a 95% confidence since

- $2 * \log_2(\ n(.95, .01)\ ) = 16.4$
- $2 * \log_2(\ n(.95, .03)\ ) = 13.2$

Just to answer the obvious next question, we have tested a dumbed-down version of SNEAK that works by doing 20 random selections (and no binary chop). It was not successful (very large variances on the output, median results many times worse than those seen in Fig. 7). Clearly, a successful optimizer needs to do more than just 20 pokes into the dark. We conjecture that SNEAK works so well because, the models' goal values are closely associated with the controllable decisions. By recursively clustering the space of decisions, we are effectively imposing a linear sort on the goal values. Hence we are performing something analogous to a stochastic version of Hoare's quickselect algorithm (albeit, evaluating fewer examples).

8 RELATED WORK

With the mathematical support, we say that the results reported in this paper are not completely unreasonable.

---

7. Since every time we find a new solution, we stop PicoSAT finding it again. This requires negating the latest solution, adding it back into the models, the re-running the algorithm from scratch.

8. Quickselect uses a similar approach to quicksort (choosing one items as a pivot and partitioning the data in two based on the pivot) but instead of recursing on both sides, quickselect only recurses into the better side [87].

Also (again, reasoning very optimistically) the maths also suggests that prior research (that did not use some quickselect method) may have been needlessly complicated a large number of other optimization tasks. That is a speculation that we will need to test, in other domains, in future work.

In other work, the configuration of product lines (e.g. our SCRUM model) has been extensively explored in the software configuration literature [88]. Based on our work in that area [89, 90], we were motivated to explore a different approach. In the configuration literature, models are evaluated by expensive and time-consuming experiments where automatic tools score configurations via functional requirements such as runtime or memory usage. Functional requirements can be assessed automatically (e.g. by running the system and measuring the performance variable of interest). SNEAK arose out of considerations on what happens when the configurations are evaluated on more subjective non-functional requirements (e.g. subjective preferences on how to configure a SCRUM project). In that case, we cannot ask a human to evaluate 100s to 1000s of different and complex configurations (due to cognitive overload). Other methods are required (hence, this paper).

Another area of related work is semi-supervised learning. In that approach conclusions are drawn from labels extrapolating from a small number of initial labels [91]. For example, *GMM with expectation-maximization* algorithms [92] uses a Gaussian mixture model to cluster the data (and uses those clusters to label the data). Further, *label propagation algorithms* [93] guess labels using a majority vote across the labels seen in nearby examples (or clusters). Label propagation algorithms never update their old labels; on the other hand *label spreading* algorithms [94] update old labels using feedback from subsequent labeling. The label spreading algorithm iterates on a similarity matrix between examples and normalizes the edge weights by computing the normalized graph of the Laplacian. Most methods described within the literature on semi-supervised learning deal with labeling problems where we are working towards a single-goal. In this paper, we explore systems with multiple goals.

9 CONCLUSION

SNEAK addresses a common problem seen in a wide range of iSBSE tools. Specifically, when AI tools find millions of solutions, human preferences must be applied to find which solution are acceptable for the current project. In theory, iSBSE methods can offer assistance for this problem, but the current generation of iSBSE methods ask too many questions to humans. Hence, they many not scale due to human cognitive overload issues.

If the problem is "too many possibilties" then a solution might be to "prune the possibilties". Here we apply some data mining methods paired with recursive biclustering, and entropy feature weighting. To the best of our knowledge, this has not been explored before in the iSBSE literature.

Pre-experimentally, we were worried that pruning the regions which violate user preferences could also cull options that were essential to the optimization. For the

models studied here, this tuned out not to be the case. Compared to the prior state-of-the-art (Araújo et al. [6]), SNEAK was at least as effective and much less expensive. As shown in Fig. 6, SNEAK asks orders of magnitude fewer questions than Araújo et al. In all our runs, SNEAK was able to find solutions within 1% to 3% of the best solutions seen in our samples. Araújo et al., on the other hand performed far worse (see Fig. 7) in the ten complex models (i.e. SCRUM, BILLING and the randomly generated ones). For simple models, SNEAK performed similarly to the baseline.

Also, the results found by SNEAK nearly always achieved their acceptance goal by the expert community. In all of the experiments but one (where the Random selection actually selected a solution with better d2h than SNEAK) the human experts preferred the solution provided by SNEAK.

Further, while 100% of SNEAK's solutions are valid, very few of the solutions found by Araújo et al. satisfied model constraints on the complex SPL models. (see Fig. 4).

Lastly, as seen in the discussion on **RQ3** we can estimate the human oracle cost of running SNEAK is far faster than using the other methods studied here. For example, under the assumption that a human oracle requires 7.5 seconds per attribute (to answer questions about different solutions) then compared to Araújo et al.:

- From POM3, SNEAK is 18 times faster;
- For SCRUM, SNEAK is 350 times faster;
- For our largest model with 1000 nodes, SNEAK is 4200 times faster.

Given the results, and as stated in the introduction, our conclusion is that SNEAK is a candidate baseline tool against which future iSBSE tools could be compared. Also we suggest to the community that it could be insightful to explore the core hypothesis of this work:

> The **SNEAK hypothesis**: *When optimizing a model using human-in-the-loop, data mining methods (that recurses into divisions of the data) do better than evolutionary methods (that evolve multiple candidate solutions over many generations)*

REFERENCES

[1] M. Mendonca, M. Branco, and D. Cowan, "Splot: software product lines online tools," in *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, 2009, pp. 761–762.

[2] A. Biere, "Picosat essentials," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 4, no. 2-4, pp. 75–97, 2008.

[3] A. Ramirez, J. R. Romero, and C. L. Simons, "A systematic review of interaction in search-based software engineering," *IEEE Transactions on Software Engineering*, vol. 45, no. 8, pp. 760–781, 2018.

[4] J. Chen, V. Nair, R. Krishna, and T. Menzies, ""sampling" as a baseline optimizer for search-based software engineering," *IEEE Transactions on Software Engineering*, vol. 45, no. 6, pp. 597–614, 2018.

[5] E. Zitzler and S. Künzli, "Indicator-based selection in multiobjective search," in *International conference on parallel problem solving from nature*. Springer, 2004, pp. 832–842.

[6] A. A. Araújo, M. Paixao, I. Yeltsin, A. Dantas, and J. Souza, "An architecture based on interactive optimization and machine learning applied to the next release problem," *Automated Software Engineering*, vol. 24, no. 3, pp. 623–671, 2017.

[7] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[8] T. do Nascimento Ferreira, A. A. Araújo, A. D. B. Neto, and J. T. de Souza, "Incorporating user preferences in ant colony optimization for the next release problem," *Applied Soft Computing*, vol. 49, pp. 1283–1296, 2016.

[9] A. Dantas, I. Yeltsin, A. A. Araújo, and J. Souza, "Interactive software release planning with preferences base," in *International Symposium on Search Based Software Engineering*. Springer, 2015, pp. 341–346.

[10] P. Tonella, A. Susi, and F. Palma, "Interactive requirements prioritization using a genetic algorithm," *Information and software technology*, vol. 55, no. 1, pp. 173–187, 2013.

[11] F. Palma, A. Susi, and P. Tonella, "Using an smt solver for interactive requirements prioritization," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011, pp. 48–58.

[12] L. Troiano, C. Birtolo, and R. Armenise, "A validation study regarding a generative approach in choosing appropriate colors for impaired users," *SpringerPlus*, vol. 5, no. 1, pp. 1–26, 2016.

[13] A. E. El Yamany, M. Shaheen, and A. S. Sayyad, "Opti-select: An interactive tool for user-in-the-loop feature selection in software product lines," in *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools-Volume 2*, 2014, pp. 126–129.

[14] C. L. Simons, J. Smith, and P. White, "Interactive ant colony optimization (iaco) for early lifecycle software design," *Swarm Intelligence*, vol. 8, no. 2, pp. 139–157, 2014.

[15] C. L. Simons and I. C. Parmee, "Elegant object-oriented software design via interactive, evolutionary computation," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1797–1805, 2012.

[16] C. L. Simons, I. C. Parmee, and R. Gwynllyw, "Interactive, evolutionary search in upstream object-oriented class design," *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 798–816, 2010.

[17] C. L. Simons and I. C. Parmee, "An empirical investigation of search-based computational support for conceptual software engineering design," in *2009 IEEE International Conference on Systems, Man and Cybernetics*. IEEE, 2009, pp. 2503–2508.

[18] J. C. Quiroz, S. J. Louis, A. Shankar, and S. M. Dascalu, "Interactive genetic algorithms for user interface design," in *2007 IEEE congress on evolutionary computation*. IEEE, 2007, pp. 1366–1373.

[19] N. Monmarché, G. Nocent, M. Slimane, G. Venturini, and P. Santini, "Imagine: a tool for generating html style sheets with an interactive genetic algorithm based on genes frequencies," in *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 99CH37028)*, vol. 3. IEEE, 1999, pp. 640–645.

[20] S. Axelsson, D. Baca, R. Feldt, D. Sidlauskas, and D. Kacan, "Detecting defects with an interactive code review tool based on visualisation and machine learning," in *the 21st International Conference on Software Engineering and Knowledge Engineering (SEKE 2009)*, 2009.

[21] B. Marculescu, S. Poulding, R. Feldt, K. Petersen, and R. Torkar, "Tester interactivity makes a difference in search-based software testing: A controlled experiment," *Information and Software Technology*, vol. 78, pp. 66–82, 2016.

[22] B. Marculescu, R. Feldt, R. Torkar, and S. Poulding, "An initial industrial evaluation of interactive search-based testing for embedded software," *Applied Soft Computing*, vol. 29, pp. 26–39, 2015.

[23] B. Marculescu, R. Feldt, and R. Torkar, "Objective re-weighting to guide an interactive search based software testing system," in *2013 12th International Conference on Machine Learning and Applications*, vol. 2. IEEE, 2013, pp. 102–107.

[24] Y. Lin, X. Peng, Y. Cai, D. Dig, D. Zheng, and W. Zhao, "Interactive and guided architectural refactoring with search-based recommendation," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016, pp. 535–546.

[25] M. W. Mkaouer, M. Kessentini, S. Bechikh, K. Deb, and M. Ó Cinnéide, "Recommendation system for software refactoring using innovization and interactive dynamic optimization," in *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, 2014, pp. 331–336.

[26] M. W. Mkaouer, "Interactive code smells detection: An initial investigation," in *International Symposium on Search Based Software Engineering*. Springer, 2016, pp. 281–287.

[27] B. Amal, M. Kessentini, S. Bechikh, J. Dea, and L. B. Said, "On the use of machine learning and search-based software engineering for ill-defined fitness function: a case study on software refactoring," in *International Symposium on Search Based Software Engineering*. Springer, 2014, pp. 31–45.

[28] A. Ghannem, G. El Boussaidi, and M. Kessentini, "Model refactoring using interactive genetic algorithm," in *International Symposium on Search Based Software Engineering*. Springer, 2013, pp. 96–110.

[29] J. Wang, X. Peng, Z. Xing, and W. Zhao, "Improving feature location practice with multi-faceted interactive exploration," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 762–771.

[30] G. Bavota, F. Carnevale, A. De Lucia, M. Di Penta, and R. Oliveto, "Putting the developer in-the-loop: an interactive ga for software re-modularization," in *International Symposium on Search Based Software Engineering*. Springer, 2012, pp. 75–89.

[31] M. S. Feather and T. Menzies, "Converging on the optimal attainment of requirements," in *Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on*. IEEE, 2002, pp. 263–270.

[32] T. Menzies, O. Elrawas, J. Hihn, M. Feather, R. Madachy, and B. Boehm, "The business case for automated software engineering," in *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '07. New York, NY, USA: ACM, 2007, pp. 303–312.

[33] W. Fu, T. Menzies, and X. Shen, "Tuning for software analytics: Is it really necessary?" *Information and Software Technology*, vol. 76, pp. 135–146, 2016.

[34] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Computing Surveys (CSUR)*, vol. 45, no. 1, pp. 1–61, 2012.

[35] S. L. Morgan and S. N. Deming, "Simplex optimization of analytical chemical methods," *Analytical chemistry*, vol. 46, no. 9, pp. 1170–1181, 1974.

[36] S.-J. Huang and N.-H. Chiu, "Optimization of analogy weights by genetic algorithm for software effort estimation," *Information and software technology*, vol. 48, no. 11, pp. 1034–1045, 2006.

[37] K. V. Kumar, V. Ravi, M. Carr, and N. R. Kiran, "Software development cost estimation using wavelet neural networks," *Journal of Systems and Software*, vol. 81, no. 11, pp. 1853–1867, 2008.

[38] S.-J. Huang, N.-H. Chiu, and L.-W. Chen, "Integration of the grey relational analysis with genetic algorithm for software effort estimation," *European Journal of Operational Research*, vol. 188, no. 3, pp. 898–909, 2008.

[39] N.-H. Chiu and S.-J. Huang, "The adjusted analogy-based software effort estimation based on similarity distances," *Journal of Systems and Software*, vol. 80, no. 4, pp. 628–640, 2007.

[40] L. L. Minku and X. Yao, "Software effort estimation as a multiobjective learning problem," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 22, no. 4, p. 35, 2013.

[41] F. Sarro, A. Petrozziello, and M. Harman, "Multi-objective software effort estimation," in *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*. IEEE, 2016, pp. 619–630.

[42] A. L. Oliveira, P. L. Braga, R. M. Lima, and M. L. Cornélio, "GA-based method for feature selection and parameters optimization for machine learning regression applied to software effort estimation," *information and Software Technology*, vol. 52, no. 11, pp. 1155–1166, 2010.

[43] L. L. Minku and X. Yao, "An analysis of multi-objective evolutionary algorithms for training ensemble models based on different performance measures in software effort estimation," in *Proceedings of the 9th international conference on predictive models in software engineering*. ACM, 2013, p. 8.

[44] F. Sarro, F. Ferrucci, and C. Gravino, "Single and multi objective genetic programming for software development effort estimation," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, ser. SAC '12. New York, NY, USA: ACM, 2012,

pp. 1221–1226.

[45] J. Chen, V. Nair, R. Krishna, and T. Menzies, ""Sampling" as a baseline optimizer for search-based software engineering," *IEEE Transactions on Software Engineering*, 2018.

[46] J. Chen, V. Nair, and T. Menzies, "Beyond evolutionary algorithms for search-based software engineering," *Information and Software Technology*, vol. 95, pp. 281–294, 2018.

[47] G. Mathew, T. Menzies, N. A. Ernst, and J. Klein, ""short" er reasoning about larger requirements models," in *2017 IEEE 25th International Requirements Engineering Conference (RE)*. IEEE, 2017, pp. 154–163.

[48] X. Du, X. Yao, Y. Ni, L. L. Minku, P. Ye, and R. Xiao, "An evolutionary algorithm for performance optimization at software architecture level," in *Evolutionary Computation (CEC), 2015 IEEE Congress on*. IEEE, 2015, pp. 2129–2136.

[49] M. H. Ali, B. A. D. Al Mohammed, A. Ismail, and M. F. Zolkipli, "A new intrusion detection system based on fast learning network and particle swarm optimization," *IEEE Access*, vol. 6, pp. 20 255–20 261, 2018.

[50] A. S. Sadiq, B. Alkazemi, S. Mirjalili, N. Ahmed, S. Khan, I. Ali, A.-S. K. Pathan, and K. Z. Ghafoor, "An efficient ids using hybrid magnetic swarm optimization in wanets," *IEEE Access*, vol. 6, pp. 29 041–29 053, 2018.

[51] S. Zhong, T. M. Khoshgoftaar, and N. Seliya, "Analyzing software measurement data with clustering techniques," *IEEE Intelligent Systems*, vol. 19, no. 2, pp. 20–27, 2004.

[52] W. Fu, T. Menzies, and X. Shen, "Tuning for software analytics: Is it really necessary?" *Information and Software Technology*, vol. 76, pp. 135–146, 2016.

[53] Y. Liu, T. M. Khoshgoftaar, and N. Seliya, "Evolutionary optimization of software quality modeling with multiple repositories," *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 852–864, 2010.

[54] A. Agrawal and T. Menzies, "Is better data better than better data miners?: on the benefits of tuning smote for defect prediction," in *Proceedings of the 40th International Conference on Software Engineering*. ACM, 2018, pp. 1050–1061.

[55] A. B. De Carvalho, A. Pozo, and S. R. Vergilio, "A symbolic fault-prediction model based on multiobjective particle swarm optimization," *Journal of Systems and Software*, vol. 83, no. 5, pp. 868–882, 2010.

[56] F. Sarro, S. Di Martino, F. Ferrucci, and C. Gravino, "A further analysis on the use of genetic algorithm to configure support vector machines for inter-release fault prediction," in *Proceedings of the 27th annual ACM symposium on applied computing*. ACM, 2012, pp. 1215–1220.

[57] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Automated parameter optimization of classification techniques for defect prediction models," in *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*. IEEE, 2016, pp. 321–332.

[58] R. B. Abdessalem, S. Nejati, L. C. Briand, and T. Stifter, "Testing vision-based control systems using learnable evolutionary algorithms," in *Proceedings of the 40th International Conference on Software Engineering*, ser. ICSE '18. New York, NY, USA: ACM, 2018, pp. 1016–1026.

[59] V. Nair, Z. Yu, T. Menzies, N. Siegmund, and S. Apel, "Finding faster configurations using flash," *IEEE Transactions on Software Engineering*, vol. 46, no. 7, pp. 794–811, 2018.

[60] V. Nair, T. Menzies, N. Siegmund, and S. Apel, "Using bad learners to find good configurations," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 257–267.

[61] W. Fu and T. Menzies, "Easy over hard: A case study on deep learning," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, 2017, pp. 49–60.

[62] A. Agrawal, W. Fu, and T. Menzies, "What is wrong with topic modeling? and how to fix it using search-based software engineering," *Information and Software Technology*, vol. 98, pp. 74–88, 2018.

[63] C. Treude and M. Wagner, "Predicting good configurations for github and stack overflow topic models," in *Proceedings of the 16th International Conference on Mining Software Repositories*, ser. MSR '19. Piscataway, NJ, USA: IEEE Press, 2019, pp. 84–95.

[64] A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 522–531.

[65] M. Skoglund and P. Runeson, "Reference-based search strategies in systematic reviews," in *13th International Conference on Evaluation and Assessment in Software Engineering (EASE) 13*, 2009, pp. 1–10.

[66] K. R. Felizardo, E. Mendes, M. Kalinowski, É. F. Souza, and N. L. Vijaykumar, "Using forward snowballing to update systematic reviews in software engineering," in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 2016, p. 53.

[67] R. Valerdi, "Heuristics for systems engineering cost estimation," *IEEE Systems Journal*, vol. 5, no. 1, pp. 91–98, 2010.

[68] M. R. Shackelford, "Implementation issues for an interactive evolutionary computation system," in *Proceedings of the 9th annual conference companion on Genetic and evolutionary computation*, 2007, pp. 2933–2936.

[69] H. Takagi, "Interactive evolutionary computation: System optimization based on human subjective evaluation," in *IEEE International Conference on Intelligent Engineering Systems*, vol. 1998, 1998, pp. 17–19.

[70] ——, "Active user intervention in an ec search," in *Proceedings of the Fifth Joint Conference on Information Sciences, JCIS 2000*, 2000, pp. 995–998.

[71] ——, "Interactive evolutionary computation: Fusion of the capabilities of ec optimization and human evaluation," *Proceedings of the IEEE*, vol. 89, no. 9, pp. 1275–1296, 2001.

[72] F.-C. Hsu and P. Huang, "Providing an appropriate search space to solve the fatigue problem in interactive evolutionary computation," *New Generation Computing*, vol. 23, no. 2, pp. 115–127, 2005.

[73] R. dos Santos Rocha and M. Fantinato, "The use of software product lines for business process management: A systematic literature review," *Information and Software Technology*, vol. 55, no. 8, pp. 1355–1373, 2013.

[74] K. Pohl, G. Böckle, and F. J. van Der Linden, *Software product line engineering: foundations, principles and techniques*. Springer Science & Business Media, 2005.

[75] C. Paul and N. Linda, "Software product lines: practices and patterns," 2001.

[76] K. Schwaber and J. Sutherland, "The scrum guide," *Scrum Alliance*, vol. 21, p. 19, 2011.

[77] T. Ngo, "Data mining: practical machine learning tools and technique, by ian h. witten, eibe frank, mark a. hall," *ACM Sigsoft Software Engineering Notes*, vol. 36, no. 5, pp. 51–52, 2011.

[78] T. Wagner, N. Beume, and B. Naujoks, "Pareto-, aggregation-, and indicator-based methods in many-objective optimization," in *EMO*, 2006.

[79] 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 *2013 35Th international conference on software engineering (ICSE)*. IEEE, 2013, pp. 492–501.

[80] T. Menzies and J. Richardson, "Xomo: Understanding development options for autonomy," in *COCOMO forum*, vol. 2005, 2005.

[81] B. Boehm and R. Turner, "Using risk to balance agile and plan-driven methods," *Computer*, vol. 36, no. 6, pp. 57–66, 2003.

[82] ——, *Balancing agility and discipline: A guide for the perplexed*. Addison-Wesley Professional, 2003.

[83] D. Port, A. Olkov, and T. Menzies, "Using simulation to investigate requirements prioritization strategies," in *2008 23rd IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 2008, pp. 268–277.

[84] C. Kastner, T. Thum, G. Saake, J. Feigenspan, T. Leich, F. Wielgorz, and S. Apel, "Featureide: Aalerdi [67] reports that his panels of human experts required three meetings (three hours each) to reach convergence on the influence of 10 variables on 10 examples (in the domain of cost estimation). tool framework for feature-oriented software development," in *2009 IEEE 31st International Conference on Software Engineering*, 2009, pp. 611–614.

[85] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 1–10. [Online]. Available: https://doi.org/10.1145/1985793.1985795

[86] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. New York: Addison-Wesley, 1989.

[87] C. A. R. Hoare, "Algorithm 65: Find," *Commun. ACM*, vol. 4, no. 7, p. 321–322, jul 1961. [Online]. Available:

https://doi.org/10.1145/366622.366647

[88] J. A. Galindo, D. Benavides, P. Trinidad, A.-M. Gutiérrez-Fernández, and A. Ruiz-Cortés, "Automated analysis of feature models: Quo vadis?" *Computing*, vol. 101, no. 5, pp. 387–433, 2019.

[89] V. Nair, R. Krishna, T. Menzies, and P. Jamshidi, "Transfer learning with bellwethers to find good configurations," *CoRR*, vol. abs/1803.03900, 2018.

[90] J. Chen, V. Nair, and T. Menzies, "Beyond evolutionary algorithms for search-based software engineering," *Information and Software Technology*, 2017.

[91] X. J. Zhu, "Semi-supervised learning literature survey," 2005.

[92] J. Hui, "Machine learning — expectation-maximization algorithm (em)," 2019. [Online]. Available: https://tiny.cc/jhui

[93] X. Zhu and Z. Ghahramani, "Learning from labeled and unlabeled data with label propagation," Tech. Rep., 2002.

[94] D. Zhou, O. Bousquet, T. Lal, J. Weston, and B. Schölkopf, "Learning with local and global consistency," in *Advances in Neural Information Processing Systems*, S. Thrun, L. Saul, and B. Schölkopf, Eds., vol. 16. MIT Press, 2004. [Online]. Available: https://proceedings.neurips.cc/paper/2003/file/87682805257e619d49b8e0dfdc14affa-Paper.pdf