When the AI Listens: Developing Scalable Interactive Search for Process Lines

Andre Lustosa, Tim Menzies, *Fellow, IEEE*

Abstract—IN PROGRESS

Index Terms—...

1 Introduction

I need help with this, i don't feel like this reads well. Interactive Search Based Software Engineering (iSBSE) is a process, also known as "human-in-the-loop", where in order to achieve a higher level of trust in the final result [11] the human and the intelligent tool, have to interact in a way where the tool modifies the result or the search process through the human evaluations. This is done by inserting the human in the intelligent search process at key moments, evaluating partial solutions, or setting search parameters, making the solutions of such an approach more easily acceptable [9]. Specifically in this paper we talk about the usage of interactive search on Software Product Lines (SPLs), which is, to generate a final optimized "Product" according to predefined metrics through feature selection.

A clear problem with this approach is the automation bias that can be caused by an extensive process of human evaluations. This has been previously documented as a problem of user complacency when users become over reliant in the tool support [12] [16] [14] [19]. There are also reports that an over-reliance on repeated "human-in-the-loop" evaluations can result in user fatigue. [18]. So a strategy to reduce automation bias is focusing on cognitive load reduction [12]. This is remarkably important in the case of SPLs when the SPL is very large and the number of features evaluated is high.

The research on interactive search in SPLs is very limited. There are a myriad of papers dealing with SPLs as a simple search problem, where no human interaction is present, such as in Hierons et al. [8] where the authors used an multi-objective evolutionary optimization technique to select one optimized product according to a set of metrics. One large problem with this is with the generation of new solutions. Given a model with a large enough number of constraints, a large part of the time spent generating solutions is wasted due to invalid solutions. In one paper in particular by El Yamany et al. [7] we see human interaction being used together with an optimizer to select the best possible solution on a feature model, however no metrics of performance were given. It also uses generational techniques to come up with a solution at each interaction. We therefore turn to Araújo et al. [2] which deals with a slightly different problem, the Next Release Problem (NRP). The NRP consists in selecting which requirements will be implemented in the next release of a software system. However the authors have conjectured their architecture to be sufficiently generic to be adopted by other software engineering scenarios, specifically citing feature selection in SPLs. They use an architecture, where an evolutionary algorithm generates solutions optimizing for both the metrics and the human evaluation, and eventually collect enough data on the human to substitute him for a neural network. By using a genetic algorithm they also have the problem of generating valid solutions due to cross-tree constraints. Given a reasonable level of complexity in the SPL feature tree, it is very unlikely and time consuming to guarantee the final solution to be valid. Not only that but the method of evaluating the entire intermediate solution at once coupled with a high number of starting evaluations for the algorithm to kick in may cause automation biases due to a high cognitive load.

The premisse of this paper is that we can do better than the evolutionary approach. By utilizing dimensionality reduction, and simple datamining techniques we improve the automated generation of solutions. We show the user partial solutions for a minimized cognitive load in evaluations. We interact less by pre-filtering the important features in the SPL and ranking the features on each step. We guarantee that the final solution is valid, and we guarantee to it a certain degree of optimality.

• *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*

2 Background

Add table of prior work plus diagrams

2.1 Scope

Interactive Search Based Software Engineering (iSBSE) on its own is a domain widely studied as seen in Ramirez et al. [16]. This paper is not meant to be a detailed comparison to methods in all the different problem formulations in the iSBSE field. The main points of this paper are to demonstrate we can simplify interactive search, by applying simple dimensionality reduction techniques and reducing the number of interactions, and to apply interactive search to an area that has not been yet fully explored, which are Project Management and Agile Development. Particularly in this subarea there is one other paper however no metrics are given, being it a proposal of a possible solution [7].

Some of the basic ideas of this paper come from Chen et al. [4], where an interesting algorithm called SWAY was developed for feature selection on a Feature Model. Instead of applying evolutionary techniques they start with a very large amount of valid solutions extracted from a SAT solver. And then sample down to a better solution.

2.2 Software Product Lines and iSBSE

INCREASE SECTION specially on iSBSE needs better background on it.

There is much research in the usage of software product lines (SPLs), also alternatively known as feature models. These SPLs define a tree of features and constraints defining the multiple variations of products that can be generated from them as said by Paul and Linda [15]. A lot of this research is focused on the generation of an optimal product according to certain metrics. Process Lines are a generalization of the SPL architecture design for business process management, as seen in the extensive review by dos Santos Rocha and Fantinato [6]. In this review it is shown that there is an extensive basis for the usage of Software Product Lines to define Business Process Configuration. Containing the same design pattern and characteristics of an SPL it is clear that we can consider different Business Process Configurations as products from a Product Line, being intuitive that the usage of AI technologies on an Process Model to generate a solution is a valid approach. Therefore throughout this paper when SPL is cited you can read it as Software Process Line or Software Product Line, since the terms are interchangeable. The automatic generation of a solution, however, has the problem of trust [11], where no involvement of a human in the process can be detrimental to the trust of the overall solution. So in this paper we work to achieve the conjunction of automatic optimization and interaction with the human expert in order to solve the trust issue.

Another problem in human interaction is the over-reliance on repeated human evaluation, which leads to user fatigue. This would then lead to inconsistent results and automation biases [16]. A secondary problem that we introduce in this paper is a cognitive load from each interaction. Given a big enough model, it is unrealistic to expect a human to be able to evaluate the proposed solution as a whole multiple times. Therefore we have worked to reduce the cognitive load both in the number of interactions and reducing the information presented in each interaction.

The final problem is how to capture subjective evaluation into the system and how to take it into consideration. This evaluation, which has been refered to as the *"quality without a name"* by Alexander [1], is part of some ill-defined aspects of good systems, endorsed by the software design pattern community [20]. Which is a pointer towards the need for human interaction in the search.

2.3 Why SCRUM

SCRUM is defined as a lightweight framework that helps organizations to deal with complex problems through adaptive solutions [17]. And although by reading the SCRUM manual directly the rules are defined as unchanging. In reality it is highly adaptive to each environment and it's necessities. We then extracted a SCRUM feature model from SPLOT-Research [13] where we can observe 124 features and 2863 constraints. Being this a simplified feature model it is not hard to argue that there might be a degree of suboptimality on the usage of this methodology by development teams since there is such a large space that might make good use of optimization and search.

By using a project management feature model we also ensure to have one of the qualities that is pointed out to be a sign towards the necessity of human interaction. Being an environment that partakes in the *"quality without a name"* [1] makes it hard to adequately formulate a representation of the metrics involved in selecting rules for a development team. In these scenarios human interaction might be key for trust and acceptance of the solution.

2.4 Dimensionality Reduction

IN PROGRESS

Discuss Why it matters

Discuss K-Means and DBSCAN

Talk about instance and feature selection.

If we cluster via random projections the contrast set between them gives us row and then the feature analysis gives us columns

scholar search for cheat sheets:

- data reduction software engineering
- instance selection software engineering
- feature selection software engineering

3 METHODS

This section discusses on the methods and techniques used throughout the experiment, as well as the origins and characteristics of the feature model.

3.1 The Feature Model

The feature model shown in Fig. 1 is the one that was utilized in our experiment. Containing a total of 124 features and 2863 constraints. This model was converted from the SXFM SPLOT-Research format into the Dimacs format using the FeatureIDE [10]. The Dimacs format is a standard interface to most sat solvers making it possible for us to generate a large number of solutions initially by utilizing standard state of the art SAT Solvers.

The feature model contains many mandatory features, focusing the selections further down the leaves. However the diverse regular and cross-tree constraints create a very complex environment from which a very large amount of solutions can be generated.

It also contains generic options or features for the pre-configuration done by the business expert. There are non descriptive options of different activities for each role that can be defined by the business expert as the different tasks

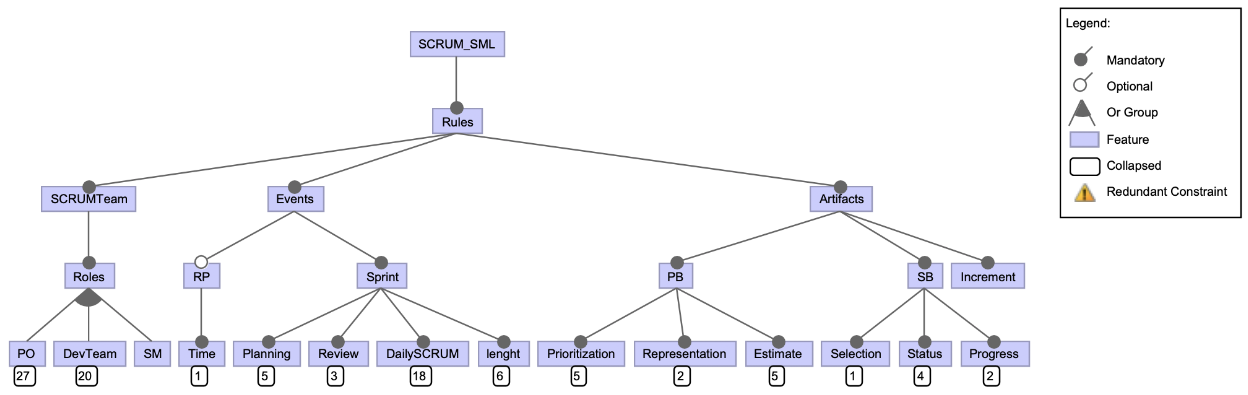

Fig. 1. The SCRUM Feature Model

he sees that role doing in order to be able to configure his own version of SCRUM.

This model is compiled as a regular SPL with none of the different characteristics present in Process Lines, therefore it was considered a good model to be used as a generalization of SPLs into the Business Process Configuration domain.

3.2 The Algorithm

Our approach starts with generating a large number of valid solutions to the given SCRUM feature model. This is done using PicoSAT [3]. PicoSAT is a state of the art python SAT solver that takes in a DIMACS formated CNF Sat file. This DIMACS file is generated from the SXFM SPLOT-Research using the featureIDE [10]. From there we have our search space of valid solutions. To each of those solutions we assign an initial weight of 1.

We also assign randomly values for each of the 124 features on the model. These values are the cost of the feature, an integer between 0 and 10. And two booleans representing whether there was a previously known problem with the feature and if it has been used before successfully. These numbers then are added up for each of the valid solutions, generating a cost, number of known problems, and number of previously used features to each solution. Since the numbers are generating once per start of run they are consistent throughout the entire search space of valid solutions.

The feature model is further enriched by a text dataset comprising of human-readable descriptions of each of the 124 features on the SCRUM Model. This dataset was manually generated collecting information from the Scrum Guide [17]

Following on the ideas set forth by [4] and previously by Chung et al. [5] in their radial kernel function. We radially separate the solutions based on the original 124 features in the dataset. This is done because as said by Chen et al. [4], when applying non radial kernels on binary spaces all the candidate solutions would be on the vertices of the D-dimensional space where D is the number of features on the model. Meanwhile applying a radial kernel forces the candidate solutions away from the vertices, giving us a better split for the separation of the different candidates.

From there we generate a radial tree, akin to a decision tree, separating and clustering the candidate solutions. Each leaf containing a parameterized N number of candidates. This tree is generated according to Algorithm 1 and Algorithm 2.

Algorithm 1 Needs a Name

```
    Input Items
    Output CTree
    Parameter Enough
    Require Func TreeNode

1: if $len(Items) < Enough$ then
2:     return $TreeNode(Items)$
3: else
4:     [w,e],[w_items, e_items] ← SPLIT(items, $\log |items|$)
5:     w_node ← NeedsAName(w_items, enough)
6:     e_node ← NeedsAName(e_items, enough)
7:     root ← $TreeNode(e, w, e\_node, w\_node)$
8:     return root
```

We also go through the solutions in order to figure out the features that actually matter to this model. This is done via a simple counting process, where we sum all the solutions together. Whatever features are always present or always absent are considered non-essential features. The features are then ranked, from most entropic to least entropic. This information is later used on our human interaction strategy.

Having generated this Clustered Tree (CTree) we then move to rank each of the rank each of the nodes. These scores are obtained by checking the difference between the east and west representatives of the tree node and multiplying that difference by the ranks of each feature. This is then divided by the total number of different features between east and west representatives. Finally we return the node with the highest score among the available ones. Which is described in Algorithm 3

Algorithm 2 SPLIT for binary decision spaces [4]

```
    Input Items
    Output [west, east], [west_items, east_items]
    Parameter TotalGroups

 1: $west \leftarrow []$
 2: $east \leftarrow []$
 3: $west\_items \leftarrow []$
 4: $east\_items \leftarrow []$
 5: $rand \leftarrow$ random item from candidates
 6: $max\_r \leftarrow -\infty$
 7: $minx\_r \leftarrow \infty$
 8: foreach $x \in Items$ do
 9:     $x.r \leftarrow \sum_{n=1}^{||x||} x\_n$
10:     $x.d \leftarrow \sum_{n=1}^{||x||} x\_n - rand\_n$
11:     if $x.r > max\_r$ then
12:         return $max\_r = x.r$
13:     if $x.r < min\_r$ then
14:         return $min\_r = x.r$
15: foreach $x \in Items$ do
16:     $x.r = \frac{x.r - min\_r}{max\_r - min\_r + 10^{-32}}$
17: $R \leftarrow \{i.r | i \in Items\}$
18: foreach $k \in R$ do
19:     $g = \{i | i.r = k\}$
20:     sort $g$ by $x.d$. $g \leftarrow x_1, x_2, ..., x_{|g|}$
21:     foreach $i \in [1, |g|]$ do
22:         $x_i.\theta \leftarrow \frac{2\pi i}{|g|}$
23: $thk \leftarrow \frac{max\_r}{TotalGroups}$
24: foreach $a \leq TotalGroups$ do
25:     $g \leftarrow \{i | (a-1)thk \leq i.r \leq (a)thk\}$
26:     sort $g$ by $g.\theta$
27:     $east \leftarrow MIN\_g$
28:     $west \leftarrow MAX\_g$
29:     $east\_items \leftarrow \{i | i \in g \ \& \ i.\theta \leq \pi\}$
30:     $west\_items \leftarrow \{i | i \in g \ \& \ i.\theta > \pi\}$
    return $[west, east], [west\_items, east\_items]$
```

Finally then we generate the human-readable questions from the differences between the best node. Chosen using the rank of each feature and also the overall difference between each half of the tree below it. In order to prevent cognitive overload we don't show the entire difference map to the human, instead, we limit it to a parameterizable number of features (N), showing the N top ranked features. A human is then inquired on which features from the subset are of more interest to him. With this information we shift the feature weights. Effectively zeroing any importance on asked features. The responses are then saved and kept aside. Knowing how many features are of relevance we have a stopping point to the algorithm, that repeats itself until it runs out of important information to ask the user. At this point we will have significantly reduced the search space, limiting it to options that are relevant to the user's opinion.

3.3 The Optimization Goal

After interacting with the human and restricting the search space. This is now a very simple search problem. We defined for this specific model the minimization equation 1.

Algorithm 3 Ranking

```
    Input Root, FeatureRanks
    Output BestNode
    Require Func WestEastDiff
// Returns the difference array between east and west representatives
 1: $largest \leftarrow -\infty$
 2: $q \leftarrow [[root, 1]]$
 3: while $q \neq \emptyset$ do
 4:     $p \leftarrow q.pop()$
 5:     if p $\neq$ Leaf then
 6:         node $\leftarrow$ p[0]
 7:         diff $\leftarrow$ WestEastDiff(node)
 8:         $node.score \leftarrow (\frac{\sum_{n=1}^{||node||} diff[n]*rank[n])*node.weight}{\sum n=1^{||node||} diff[n]})$
 9:         if $node.score > largest$ then
10:             $largest \leftarrow node.score$
11:     if $node.west\_node \neq \emptyset$ then
12:         $q \leftarrow [node.west\_node, p[1] + 1]$
13:     if $node.east\_node \neq \emptyset$ then
14:         $q \leftarrow [node.east\_node, p[1] + 1]$
15: return largest
```

Equation 1 Minimization Equation for SCRUM Model

$$f(x) = |C|^2 + |D|^2 + |(100 - HPFS)|^2 + |(124 - P)|^2 \tag{1}$$

Where C is the cost of the solution, D is the number of known defects, HPFS is the percentage of human prioritized features selected in the solution, and P is the number of features successfully used in previous iterations. Where all values have been normalized between 0 and 1.

With this equation we are able to search and select the most appropriate solution at hand. We do this by plugging the available solutions

3.4 The Baseline

The baseline algorithm taken from Araújo et al. [2] is defined as a 3 component architecture. An interactive genetic algorithm, an interactive module to interface with the user, and a learning model. The interactive genetic algorithm is responsible for the optimization process, where candidate solutions are evaluated through the interactive module and, when required, by the learning model. For a given individual it's fitness value is calculated according to an equation considering the data and user implicit and explicit preference. The algorithm starts with a user setup of the optimization goal, as in the percentile of fitness it wishes to achieve 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. After the human interactions the model takes the part of the human continuing the process until the goal is reached. This process can be accurately described by Figure 2. Where we can see that until the number of human selected interactions is through the Learning Model builds a dataset to learn from. After which the Model is trained and steps in in lieu of the

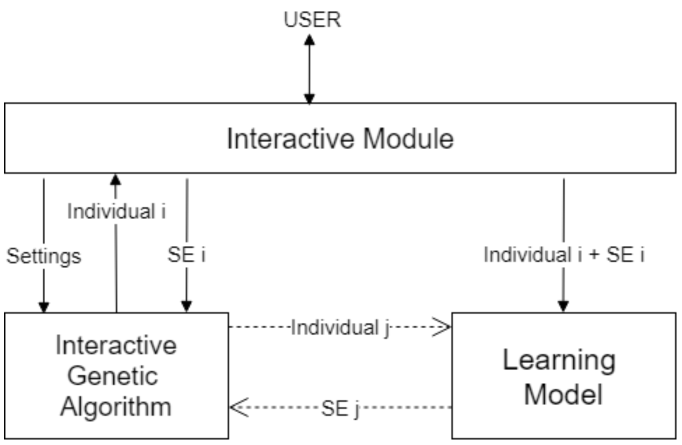

Fig. 2. Note: Conceptual architecture overview. Adapted from Araújo, A. A., Paixao, M., Yeltsin, I., Dantas, A., & Souza, J. (2017). "An architecture based on interactive optimization and machine learning applied to the next release problem." *Automated Software Engineering*, 24(3), 623-671.

human. Their algorithm shows the user the entire individual for evaluation, which is then given a numerical score. This score is included in the formula for calculating the fitness of an individual. Since this paper is an architecture we have reproduced it using a customized genetic algorithm that assesses the fitness of the solutions based on the user or model scores, and generates new solutions through mutation and crossover. We have also chosen the same model as referenced in the paper, a simple multilayered perceptron neural net with default parameters.

Equation 2 Fitness Equation for SCRUM Model

$$fit(x) = |C|^2 + |D|^2 + |(100 - SE)|^2 + |(124 - P)|^2 \quad (2)$$

The fitness equation used had to be adapted from Equation 1, now we ask our users to assess the given solution with a number between 0 and 100 as seen in Equation 2. Where SE is the subjective evaluation given by the user. The two equations are very similar and should return similar results in terms of minimization percentiles.

4 Results

The experiments were set up to be run with the usage of an oracle in lieu of an user. In both our algorithm and the baseline the oracle was designed in a way to retain memory of past decisions in order to maintain consistency along each single run.

In our algorithm the oracle keeps a track of the features it previously selected. And for each interaction it does a random selection where it favors nodes containing those features. This allows for a possibility of the oracle contradicting itself, which is a real possibility when we include users in the loop. Also allowing for different paths to be explored in the CTree. The experiment was then run for our algorithm 40 times, 20 times initializing it with 10.000 valid solutions and 20 times initializing it with 100.000 solutions, in all runs the number of features shown to a user was limited at 3 per side.

Interactions	HPFS	Score
5.0	66.66	0.01
5.0	62.5	0.01
6.0	54.54	0.01
6.0	44.44	0.01
6.0	50.0	0.01
6.0	54.54	0.01
5.0	62.5	0.01
6.0	50.0	0.01
6.0	50.0	0.01
5.0	66.66	0.01
5.0	66.66	0.01
5.0	62.5	0.01
5.0	66.66	0.01
5.0	66.66	0.01
5.0	66.66	0.01
6.0	62.5	0.015
6.0	62.5	0.01
5.0	62.5	0.01
6.0	66.66	0.01
6.0	75.0	0.015

TABLE 1
Experimental Results on 10.000 Solutions

Interactions	HPFS	Score
9.0	60.0	0.001
9.0	66.6	0.0015
9.0	66.6	0.001
9.0	66.6	0.001
9.0	71.4	0.001
9.0	70.0	0.001
9.0	66.6	0.001
9.0	54.5	0.001
9.0	63.6	0.001
9.0	70.0	0.001
9.0	61.5	0.001
9.0	66.6	0.0015
9.0	60.0	0.001
9.0	55.5	0.001
9.0	72.7	0.001
9.0	55.5	0.001
9.0	66.6	0.001
9.0	58.33	0.001
9.0	66.6	0.001
9.0	58.33	0.001

TABLE 2
Experimental Results on 100.000 Solutions

As seen on table 1 although we were dealing with a dataset of 10.000 solutions the number of interactions is very limited, averaging on 5.5 human interactions. Which demonstrates that we can achieve a solution with a very small number of interactions. The HPFS or the percentage of human prioritized features that are present in the final solution averaged 62.5% demonstrating that although the number of interactions is small we can still achive a solution with high satisfiability for the end user. Finally when applying Equation 1 to all the solutions in the dataset, we obtain the percentile of the final solution, which is the bottom 2% showing that our final solution, given it is a minimization problem, is very close to optimal.

The results from Table 1 are very similar to those in Table 2. We can see that a 10 times increase on the size of the original dataset does not dramatically increase the number of interactions, and also does not affect the HPFS.

While selecting the final solution however and calculating the percentile we find our solutions to be consistently in the bottom 0.2% demonstrating once again that the size of the dataset does not affect the optimality in the final result.

CPU-resources were not an issue either, when measuring the time it takes for each run we have realized that most of the CPU-time of the algorithm is dedicated to the generation of the initial dataset. Generating 10.000 solutions takes about a minute and 100.000 solutions takes between 10 and 15 minutes. The runtime of the search algorithm however averaged a minute for the 10.000 dataset and 2 minutes for the 100.000 dataset.

We have also done a single trial run on a initial dataset containing 1.000.000 solutions. The generation of such a dataset took 11 hours and the algorithm finished running in 10 minutes. It had 11 interactions and the HPFS and Score showed similar results to the 10.000 and 100.000 solutions runs with a 67.5 HPFS and 0.00015 score.

4.1 Baseline Results

In the baseline algorithm, the oracle is initialized with a set of values for each feature in the SCRUM feature model which are then normalized to add up to 100. When given a candidate solution, the oracle then calculates the score with these values.

Initially we set the architecture with the exact same initial non interactive genetic algorithm to create the initial population. We ran this setup 10 times with 10 interactions and a setting where the final solution should be better than 98% of the previously generated solutions. This brought us to our first issue. With such a low volume of interactions the learning model did not converge, and newly generated solutions were randomized, changing the scores from better to worse. We then increased the number of interactions until we found the necessity of 50 minimal interactions and repeated the runs, which all gave us a converging multilayered perceptron, however for every one of the runs, the final solution was not a valid solution when run through a SAT-Checker due to the complexity of the feature model and it's cross-tree constraints.

We then modified the baseline algorithm, initializing it with a population of valid solutions from PicoSAT. This then was run 10 times with the same settings as before, 50 interactions and a final solution better than 95% of the previously generated solutions. Out of all the runs, however, none generated a valid solution in the end. The solutions however, achieved scores averaging the lowest 2nd percentile.

The runtime of this baseline algorithm was slightly slower than ours averaging 20 minutes per run. However considering placing a user in front of a high volume of interactions should increase this time by a large margin.

Another issue we had was when considering the presentation to the user. Given our feature model containing 124 features, presenting a candidate solution to the user means it has to score and analyze a complete SCRUM model. This coupled with the high number of interactions, could cause a high cognitive overload potentially leading to automation biases in the final solution.

One thing that could be done to extract a valid solution is to apply checks to each mutated solution, discarding the ones that are invalid and generating new ones. However when applying this technique one single run of the baseline algorithm took 8 hours.

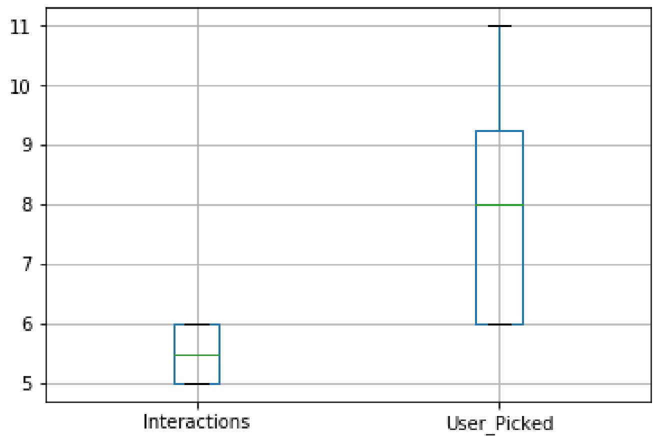

Fig. 3. Stats on Interactions 10k

4.2 Stats

We have also performed statistical analysis on several of the collected information throughout our runs of the algorithm. Specifically we have ran a Scott-Knott test on these variables. On Table X **(include xiao like table for the scott knott and remove images)** we can see that when dealing with interactions in both the 10.000 and 100.000 cases the algorithm is rather stable. With the average of human picked features being proportional to the number of interactions required. However it is to be noted that given the set number of at most 6 features shown to the user at a time, where the user would at maximum select three of those, the algorithm is effective in diminishing the numbers of features shown with each interaction. Reaching the lowest point on one feature per side in very few interactions.

The HPFS analysis is a little less stable, however with very similar averages on both 10.000 and 100.000 cases. Delving deeper into the data we have uncovered that this is caused by the eventual selection of contradictory features by the oracle. In a specific case, the oracle had selected 3 different activities for the same role in the SCRUM model. Given the exclusive nature of that particular node we can assume that instances of this might have caused more negative results. We conjecture that by putting an expert in front of the algorithm the results in this specific metric might be higher.

Finally when analyzing the percentile of the solution we see that it is a very stable algorithm. Always ending up with a solution that best satisfies 1. This solution when compared to the rest of the dataset, including the pruned and therefore not evaluated during the selection is at the lowest percentiles on the minimization problem. And the percentile tends to decrease with the increase of the dataset, leading to believe that it is selecting the best or second best among all the original solutions.

4.3 Research Questions

RQ 1 - Can we do it cheaper?

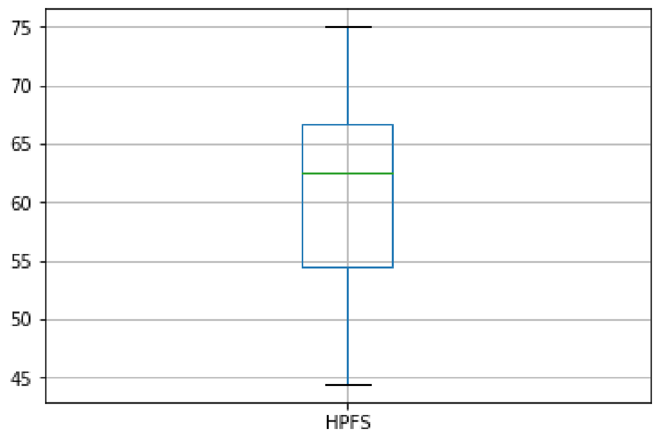

Fig. 4. Stats on HPFS 10k

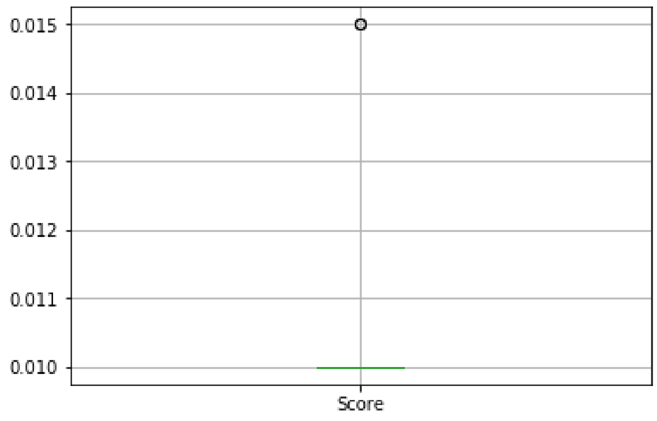

Fig. 5. Stats on Score 10k

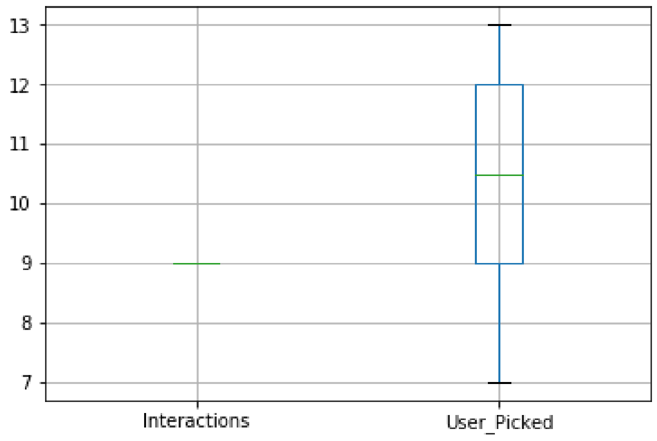

Fig. 6. Stats on Interactions 100k

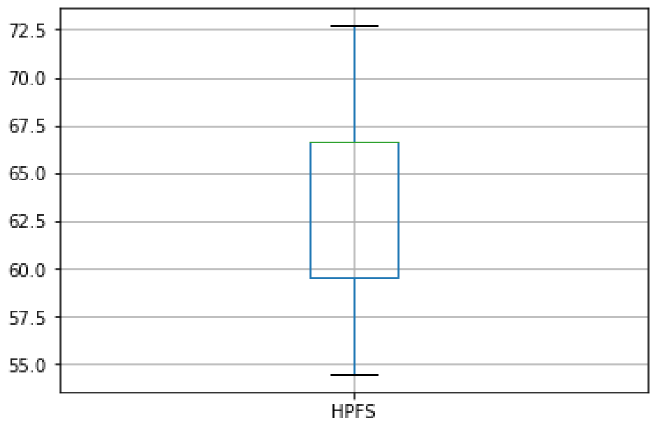

Fig. 7. Stats on HPFS 100k

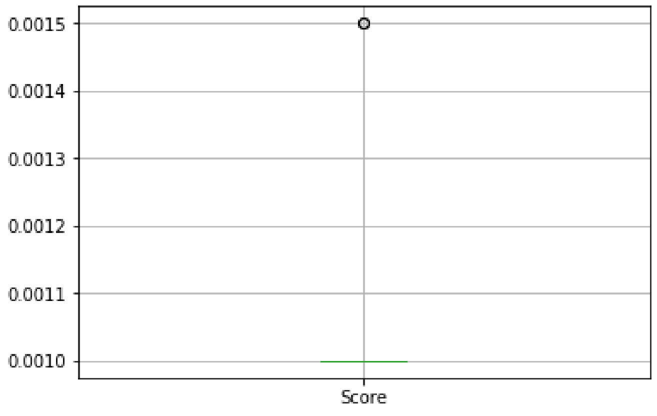

Fig. 8. Stats on Score 100k

While the baseline algorithm requires a rather large amount of interactions for this specific feature model in order for the learning model to converge and take over. Our algorithm needs a much smaller number of interactions while retaining good results. This in turn means that less user time is consumed, reducing costs. Our algorithm is also faster in terms of CPU-Time considering that the dataset of solutions has been previously generated. Finally our algorithm always generates valid solutions, without any change to it's architecture and ideas.

RQ 2 - Is the solution optimized enough?

While the evolutionary algorithm has the potential of generating a more optimized solution than ours, given the possibility of such a solution not being existent in the original dataset, the fact that it more often than not, in the SCRUM feature model, generates invalid solutions, and almost always ends up generating less solutions than the number comprised in the original datasets of our algorithm leads us to believe that the percentiles achieved in our work are representative and can be considered a high degree of optimization.

The fact that even with the pruning of over 70% of the original dataset we manage to select a solution that is better than 99% of the original dataset shows that the solution is well optimized.

RQ 3- Can we diminish cognitive load avoiding potential automation biases?

Not only our solution incurs in less interactions than the baseline, it also does not show an entire solution at once. Which greatly reduces the possibility of cognitive distress and might prevent automation biases. While this point needs to be further validated. The fact that we can achieve these results with a much diminished approach to the interactions stands to show that it is possible to work in order to reduce cognitive load.

4.4 Threats to Validity

I need help with this section
RQ 1

We only ran this algorithm on one feature model, and it would possibly be interesting to test it on larger feature models to see how it would behave. Not only that but to find different baselines to compare ourselves to.

RQ 2

There is always the risk that PicoSAT generated too similar solutions and therefore we have not completely resolved this

RQ 3

Since we ran this without actual humans and yet with a consistent oracle the qualitative aspect of cognitive load could not be assessed. Which is something to be included in future work.

5 Conclusion

IN PROGRESS
We did it. We have implemented a fast, affordable (low n^o of interactions), and performant (high scores) interactive search algorithm in the Software Product and Process Line field. Future work would be to evaluate our algorithm against different solutions, expand it to other domains, test it against different datasets, and conduct human trials on cognitive load.

Acknowledgments

The authors would like to thank...

References

[1] Christopher Alexander. *A pattern language: towns, buildings, construction*. Oxford university press, 1977.

[2] Allysson Allex Araújo, Matheus Paixao, Italo Yeltsin, Altino Dantas, and Jerffeson Souza. An architecture based on interactive optimization and machine learning applied to the next release problem. *Automated Software Engineering*, 24(3):623–671, 2017.

[3] Armin Biere. Picosat essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, 4(2-4):75–97, 2008.

[4] Jianfeng Chen, Vivek Nair, Rahul Krishna, and Tim Menzies. "sampling" as a baseline optimizer for search-based software engineering. *IEEE Transactions on Software Engineering*, 45(6):597–614, 2018.

[5] Kai-Min Chung, Wei-Chun Kao, Chia-Liang Sun, Li-Lun Wang, and Chih-Jen Lin. Radius margin bounds for support vector machines with the rbf kernel. *Neural computation*, 15(11):2643–2681, 2003.

[6] Roberto dos Santos Rocha and Marcelo Fantinato. The use of software product lines for business process management: A systematic literature review. *Information and Software Technology*, 55(8):1355–1373, 2013.

[7] Ahmed Eid El Yamany, Mohamed Shaheen, and Abdel Salam 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*, pages 126–129, 2014.

[8] Robert M Hierons, Miqing Li, Xiaohui Liu, Sergio Segura, and Wei Zheng. Sip: Optimal product selection from feature models using many-objective evolutionary optimization. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 25(2):1–39, 2016.

[9] D Jones. Programming using genetic algorithms: isn't that what humans already do, 2016.

[10] Christian Kastner, Thomas Thum, Gunter Saake, Janet Feigenspan, Thomas Leich, Fabian Wielgorz, and Sven Apel. Featureide: A tool framework for feature-oriented software development. In *2009 IEEE 31st International Conference on Software Engineering*, pages 611–614. IEEE, 2009.

[11] Glen Klien, David D Woods, Jeffrey M Bradshaw, Robert R Hoffman, and Paul J Feltovich. Ten challenges for making automation a" team player" in joint human-agent activity. *IEEE Intelligent Systems*, 19(6):91–95, 2004.

[12] David Lyell and Enrico Coiera. Automation bias and verification complexity: a systematic review. *Journal of the American Medical Informatics Association*, 24(2):423–431, 2017.

[13] Marcilio Mendonca, Moises Branco, and Donald Cowan. Splot: software product lines online tools. In *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, pages 761–762, 2009.

[14] Raja Parasuraman and Victor Riley. Humans and automation: Use, misuse, disuse, abuse. *Human factors*, 39 (2):230–253, 1997.

[15] Clements Paul and Northrop Linda. Software product lines: practices and patterns, 2001.

[16] Aurora Ramirez, Jose Raul Romero, and Christopher L Simons. A systematic review of interaction in search-based software engineering. *IEEE Transactions on Software Engineering*, 45(8):760–781, 2018.

[17] Ken Schwaber and Jeff Sutherland. The scrum guide. *Scrum Alliance*, 21:19, 2011.

[18] Mark RN Shackelford. Implementation issues for an interactive evolutionary computation system. In *Proceedings of the 9th annual conference companion on Genetic and evolutionary computation*, pages 2933–2936, 2007.

[19] Linda J Skitka, Kathleen L Mosier, and Mark Burdick. Does automation bias decision-making? *International Journal of Human-Computer Studies*, 51(5):991–1006, 1999.

[20] Goranka Zoric, Karlo Smid, et al. Design patterns: Elements of reusable object-oriented. In *Software', Erich*

Gamma, Richard Helm, Ralph Johnson and John Vlissides, Addison-Wesley. Citeseer, 1995.