SNEAK: Faster Interactive Search-based Software Engineering (using Semi-Supervised Learning)

Andre Lustosa*
alustos@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Tim Menzies,Fellow, IEEE*
timm@ieee.org
North Carolina State University
Raleigh, North Carolina, USA

ABSTRACT

When reasoning over complex models, AI tools can generate too many solutions for humans to read and understand. In this case, *interactive search-based software engineering* techniques might use human preferences to select relevant solutions (and discard the rest). Often, iSBSE methods rely on evolutionary methods that need to score thousands of mutants. Generating those scores can be overwhelming for humans or impractically slow (e.g. if some automated process required extensive CPU to computer those scores).

To address that problem, this paper introduces SNEAK, a semi-supervised learner (SSL) that uses the structure of the data to label a very small number of points, then propagates those labels over its neighbors. Whereas standard SSL addresses single goal problems (classification, regression), SNEAK is unique in that it can handle multi-goal problems. As shown by the experiments of this paper, SNEAK asks for very few labels (30, or even less) even for models with 1000 variables and state spaces of 2^{1000} possibilities. Also, the optimization solutions found in this way were within the best 1% of the entire space of solutions. Further, due to the logarithmic nature of its search, in theory, SNEAK should scale well to very large problems. Accordingly we recommend SNEAKing since, at the very least, it is a baseline architecture against which other iSBSE work can be compared. To enable that comparison, all of ours scripts are available at https://github.com/zxcv123456qwe/sneak.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

interactive search-based software engineering, data mining, model reasoning, multi-objective optimization, semi-supervised learning, cognitive load reduction

1 INTRODUCTION

AI tools are currently able to reason over very complex models, and in turn offer us thousands to millions of valid solutions to that model. Many models contain a large quantity of variables v and many more constraints. These constraints can many times introduce so many restrictions to the model that in some cases less than 1% of the available space of 2^v possible choices are legal to them. In these cases, AI tools are of great help to users, quickly finding thousands or millions of satisfying solutions to a model. For example in the SCRUM model described in this paper, a SAT Solver algorithm is capable of finding millions of valid solutions in no time.

However, this introduces a few problems. First of all, there are now too many valid solutions to choose from. Even when optimizing towards different objectives there will exist too many solutions that satisfy the optimization goals. Once these candidate solutions are generated, some form of human preference factor must be applied to find the solution better applicable to the case at hand. One way to find human preferences is through the use of *interactive search-based software engineering* [56] (iSBSE). iSBSE methods are search methods that allow humans to influence the search and learning process. A big challenge faced by most of the current iSBSE methods is *cognitive fatigue* (when humans are overwhelmed by too much information). Another problem is that when optimizing these solutions towards multiple goals, too many objective function evaluations are performed, which could potentially become very expensive in terms of computation time. For example if we are optimizing hardware design, a single evaluation requires the synthesis of a sample which could take hours or even days. One more problem is how to design an algorithm that can be generic enough to deal with a high variety of different problems, since these problems are all defined by different models with different constraints. A final problem of most methods in iSBSE is effectiveness (since many of the heuristics used by these methods end up missing important candidate solutions).

Prior work in iSBSE showed that AI tools do not need to reason over all available data. However, while greatly improving on the cognitive fatigue and generality, Lustosa et al's [36] methods end up performing too many objective function evaluations, and are also limited to binary decision spaces. Through the use of data-mining methods Lustosa et al. have been able to reduce the number of interactions by an order of magnitude compared to previous work, and also reduce the average size of these interactions by another order of magnitude. Using large dataset of candidate solutions generated from models, their methods use meta-knowledge of the space to search it eliminating solutions in disagreement with human preferences. From there their methods evaluate each surviving solution to rank them and select the best. While we agree with Lustosa et al. in the sense that the use of data mining tools seem to be the best and most versatile option for general iSBSE, when dealing with expensive evaluation functions their methods end up evaluating from 10 to 15% of the original dataset. This could prove to be a very strong limiting factor to their methods when reasoning over expensive models.

To further improve the work done by Lustosa et al. [36] in their WHUN algorithm, we have defined a semi-supervised multi-objective optimizer using data-mining technology to dramatically

*Both authors contributed equally to this research.

reduce the number of objective function evaluations. Our methods have shown to be able to generate "good enough" solutions for expensive evaluation functions evaluating less than 0.5% of the original dataset. From there we have also included a pre-processing step, to discretize any continuous variables from a model into binary ones. This is done by dividing the data into bins of ranges, according to entropy of the distribution. These two changes, represent an architecture for iSBSE. This architecture is here named SNEAK.

We as well as Lustosa et al [36] note this is a poorly explored approach to iSBSE where instead of *patching* (e.g.) a genetic algorithm with some secondary process [6, 64, 69], we *replace* the search device with data mining tools that know how to find minimal models. One reason to prefer this approach over other iSBSE tools [56] is that our methods are not specific to any kind of model. Any model that can be discretized to boolean spaces can be processed by SNEAK. Apart from that our methods are capable of dealing with expensive evaluation functions while still producing "good enough" results.

To assess this approach we have explored different models, from product line models (SCRUM, online billing) to software cost estimation models (XOMO, POM). Compared to the previous state-of-the-art [6, 36], SNEAK found solutions within a very low number of interactions and far fewer model evaluations. In our experiments SNEAK could handle very large models with less than 10 interactions, where each time we ask only three questions (median), and evaluating less than 0.5% of the original dataset of candidate solutions. The rest of this paper describes the background to the work, the SNEAK algorithm, the models used to test SNEAK, and the experimental rig used in this analysis.

2 BACKGROUND

2.1 Search-based SE

In this work, a model that describes a space of options is searched for candidate solutions. One complexity of performing this search is that there is a common trade-off between multiple competing goals. As said by Lustosa et al. [36]

- For requirements engineers, we can find the *least* cost mitigations that enable *most* requirements [22].
- For project managers, we can explore software process models to find options that deliver *more* code in *less* time with *fewer bugs* [41], or a better aligned software cost estimation model to their current productivity.
- 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 [24].

Search-based SE (SBSE) encompasses automatic methods for exploring such a trade-space [28]. Unlike more traditional optimization methods (e.g. SIMPLEX [48]), SBSE makes the more realistic assumption that to win on some goals might mean trading-off on others, instead of assuming all goals can be reached.

SBSE is a very active area of research. The work in this area explores various different issues, amongst them:

- *software configuration* [13, 49, 50];
- *project management* [11, 13, 14, 29, 30, 32, 43, 44, 51, 59, 60];
- *requirements optimization* [22, 40];
- *software design* [11, 19];
- *software security* [4, 57];
- *software quality* [1, 3, 16, 25, 35, 58, 68, 72, 76];
- *text mining* [2, 23, 53, 70];

There are many ways to do SBSE and this paper, as well as the WHUN paper [36], adopts many of the "oversampling" methods from Chen et al. [12]. As described by Lustosa et al. that work compared two approaches to optimization:

- A *traditional approach* that mutated (say) 10^2 individuals across (e.g.) 10^2 generations;
- An *oversampling approach* that filled the first generation with $(10^2)^2 = 10,000$ individuals, which are then (a) recursively clustered and then (b) pruned by removing sub-trees whose roots are dominated by their nearest neighbor.

In studies with optimizing decisions within product lines, Chen et al. were able to show that such oversampling found solutions as good, or better, than the traditional approaches. This was a natural approach for WHUN and this was also a natural approach for the SNEAK architecture. since:

- Chen et al. as well as Lustosa et al. based their empirical work on software project lines, we expand those algorithms to work within a much broader scope;
- Also, as mentioned in the introduction, our problems start with a large dataset of candidate solutions, since generating solutions is simpler than evaluating them in most cases.

2.2 Interactive Search-based SE

As stated by [36] one issue with standard SBSE methods, is that all of its conclusions are "black-box". Meaning that these algorithms will run and produce results, even if users have no understanding or input into the process. Interactive SBSE (iSBSE) is a variant of SBSE techniques that attempts to include humans into the reasoning process. More specifically, iSBSE tries to find which preferences, within a range of options, is of largest importance to users.

One question that is frequently asked towards iSBSE research is whether AI should just ignore preferences given that human knowledge and preference might end up forcing optimizers into some sub-optimal region of the data space. If we look at the papers in Table 1 we can see that has not been the general experience in the iSBSE field. The usual result being that we are able to achieve competitive optimization results when also respecting human preference. As stated by Lustosa et al. [36], there are limits to adopting all user preference, for example no matter the preference, humans still cant change gravity at sea-level on the planet earth (approx. 9.8 m/s^2). Current state-of-the-art algorithms can, respecting human preference, achieve optimization results that are within 0.5% of the best in the sample.

Lustosa et al. [36] offers a mathematical explanation to their results. In their experiments, specifically on the SCRUM model, the WHUN algorithm selects 20 preferences within a space of 128 boolean variables. That is, the acceptable space to their oracle is $2^{20}/2^{128} \approx$ a trillionth of the decision space. This means that iSBSE can still work within (*1 - 1/trillion*) % of the options.

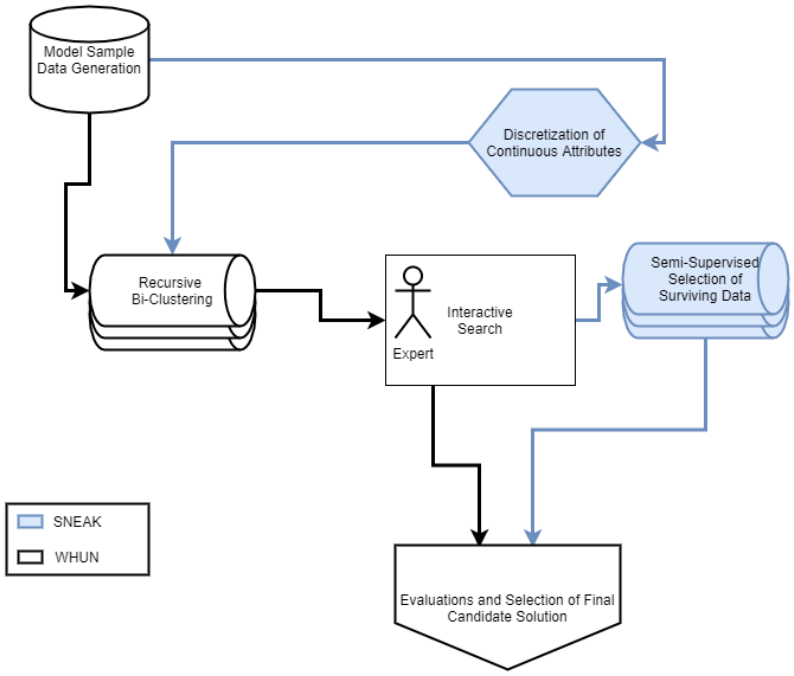

Figure 1: SNEAK Architecture vs WHUN Architecture

Paper		Technique	Citations
Interactive requirements prioritization using a genetic algorithm	[69]	Evolutionary computation	107
Interactive, evolutionary search in upstream object-oriented class design	[64]	Evolutionary computation	102
Recommendation system for software refactoring using innovization and interactive dynamic ...	[46]	Evolutionary computation	79
Putting the developer in-the-loop: an interactive GA for software re-modularization	[7]	Evolutionary computation	74
Elegant object-oriented software design via interactive, evolutionary computation	[63]	Evolutionary computation	61
Interactive genetic algorithms for user interface design	[55]	Evolutionary computation	55
Imagine: a tool for generating HTML style sheets with an interactive genetic algorithm ...	[47]	Evolutionary computation	53
Improving feature location practice with multi-faceted interactive exploration	[75]	Evolutionary computation	51
Model refactoring using interactive genetic algorithm	[27]	Evolutionary computation	49
On the use of machine learning and search-based software engineering for ill-defined fitness...	[5]	Evolutionary computation	46
An Architecture based on interactive optimization and machine learning applied to the next ...	[6]	Evolutionary computation	32
An initial industrial evaluation of interactive search-based testing for embedded software	[38]	Evolutionary computation	20
Tester interactivity makes a difference in search-based software testing: A controlled experiment	[39]	Evolutionary computation	14
OPTI-SELECT: An interactive tool for user-in-the-loop feature selection in software product lines	[20]	Evolutionary computation	13
Interactive software release planning with preferences base	[15]	Evolutionary computation	11
Objective re-weighting to guide an interactive search based software testing system	[37]	Evolutionary computation	11
An empirical investigation of search-based computational support for conceptual software ...	[62]	Evolutionary computation	9
A validation study regarding a generative approach in choosing appropriate colors for ...	[71]	Evolutionary computation	4
Interactive code smells detection: An initial investigation	[45]	Evolutionary computation	3
Interactive and guided architectural refactoring with search-based recommendation	[34]	Single-solution based	46
Interactive ant colony optimization (iACO) for early lifecycle software design	[71]	Swarm intelligence	33
Incorporating user preferences in ant colony optimization for the next release problem	[17]	Swarm intelligence	16
Using an SMT Solver for Interactive Requirements Prioritization	[52]	Exact metaheuristic	25
Preference Discovery in Large Product Lines	[36]	Exact metaheuristic	1
This paper $\Longrightarrow$		Exact metaheuristic	

Table 1: Current state of iSBSE research.

This fraction $2^{20}/2^{128}$ is also a strong motivation to why iSBSE should be an essential part of human+AI decision making. This fraction is so vanishingly small, that a fully automated algorithm has an impossibly small chance to select the options that matter to the user. Therefore iSBSE is essential to stop human preferences from being ignored in search processes.

For all these reasons, we explored the iSBSE literature. Table 1 offers an overview on the current state of iSBSE research. This table was made available on Lustosa et al.'s paper [36]. Same as Lustosa et al. we have explored a prominent paper in iSBSE:

The rest of this section will be dedicated to reviewing the iSBSE area using at least one sample paper per technique defined in Table 1

As shown by Lustosa et al. [36] many of current iSBSE methods have issues with:

- Cognitive fatigue;
- Scalability;
- and Effectiveness.

What was not brought up by Lustosa et al. is the issue of scalability when taking into consideration the cost of the evaluation function of different models. In light of these issues we will reassess some of the examples that were discussed in Lustosa et al.'s work.

As said by Lustosa et al. iSBSE methods can lead to cognitive fatigue when humans are presented with too many options repeatedly. And as we discussed previously, standard SBSE methods can generate and review millions of options. However humans are not capable of quickly and accurately assess this much material.

This is shown through Valerdi [73] whose reports showed that human experts would be unable to quickly assess and understand the effects of a small standard model in the low dozens of features, taking them 9 hours to work through the effects of 10 variables on 10 datapoints. Making the task of evaluating models with hundreds of features and millions of possible options unfeasible.

Takagi's advices [65–67] that human cognitive fatigue can be decreased by both reducing the number of interactions (I), and reducing the size of each interaction (S), are usually implemented by most of the iSBSE methods, using the decisions made along the way to ignore certain possibilities. As it was shown by Lustosa et al. these reductions were not enough when attempting to generate a more problem agnostic method.

Palma et al. [52] use a state-of-the-art constraint solver (MAX-SMT). This approach is deemed "exact" since it only explores valid solutions. This however comes at a price of extensive user interaction ($25 \leq I \leq 100$). This approach however, although very effective, has issues with cognitive fatigue and scalability. WHUN [36], while also an "exact" approach, was able to solve models 20 times as large as Palma et al. and with 2 to 5 times less interactions. Not only that, but Palma et al's methods, are single-objective optimizations, while WHUN works with multiple goals. One other problem with Palma et al's methods, is the fact that they will take every sample back to the model for evaluation, which can be a very expensive task depending on the model.

Lin et al. [34] presented a "single-solution based" method towards code refactoring tasks. Their solution will recommend refactoring "*paths*" to users, that can either accept, reject or ignore them. These interactions are then used as feedback to calculate the next recommendation. In this case S represents the number of refactorings and the size of each of them. Like Palma et al., these methods have issues with scalability and cognitive fatigue, also being single-objective.

Araújo et al. [6] use an evolutionary computation architecture. Their architecture is combined with a learner algorithm. Initially, humans are utilized to evaluate the fitness of examples. Once there are "enough" examples to train a learner, the archictecture starts to evaluate examples through the learner. Their approach will start with the setup of the optimization goal (e.g. how many generations it should run for and the number of human interactions). This setup needs to be enough to allow for the learning model to converge. The problem with this approach is that, given a model with too many constraints, it will only generate valid solutions 4% of the time in average [36].

Ferreira et al. [17] proposed a "Swarm intelligence architecture" where the user input their preferences on a particular model for each of the features of a model. The algorithm uses it to change it's search approach on the feature space and provides a user with a candidate solution. The user can choose to accept, terminating the algorithm, or reset their preferences to query for a different solution. The problem with this approach is again concerns about cognitive fatigue. Not only that but all instances of the possible solutions are evaluated and ranked.

Lustosa et al. [36] proposed a different approach. Instead of using constraint solvers or evolutionary methods, they have implemented an "exact" algoritm using a modified version of the oversampling methods by Chen et al. [12], the general workflow and architecture for the WHUN algorithm can be seen in black on Figure 1. Via a recursive binary-clustering on the features of the model, and interactions based on statistical analysis of the currently available samples the algorithm is capable of down-sampling to around 15% of the original data within less than 10 interactions on average. Within this 15% it is also possible to find results within 0.05% of the best. The problems faced by the WHUN algorithm are it's inability to deal with other datasets containing continuous features. And also the fact that once it down-samples to around 15% of the surviving data, this data needs to be evaluated and sorted within the optimization goals to select the best one.

Our methods extend Lustosa et al's work as seen in blue on Figure 1. We first apply a simple pre-processing step to discretize continuous variables into bins so that they can be used on WHUN. After it is done down-sampling the data into a better region of space, we use a semi-supervised learning algorithm to approximate data in lower dimensions, reducing the number of necessary model evaluations to reach good results.

2.3 Semi-supervised learning

Semi-supervised learners [10] train their models by combining a small amount of labeled data with a large amount of unlabeled data. This is done using assumptions on the *manifold*, *continuity* and *clustering*. The *manifold* assumption is that data can lie on an approximation manifold of much lower dimension than the original input space. Under this assumption, higher-dimensional data is approximated in a much lower dimension space, with little to no performance loss [33]. When data is distributed over just a few underlying dimensions, there are fewer ways in which examples can differ. Therefore, there is more *continuity* between nearby examples, making it unnecessary to reason separately about each example. Rather, we can *cluster* similar examples and reason about a single item per cluster.

Using those assumptions there are many ways to extrapolate from a small number of labels to a larger set of data [79]. For example, *GMM with expectation-maximization* algorithms [31] use a Gaussian mixture model to cluster the data (and use those clusters to label the data). Further, *label propagation algorithms* [78] 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 [77] 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 propose the SNEAK semi-supervised optimizer in order to down sample a large dataset within very few queries to the model. In a sense, the work by Lustosa et al. [36] can also be named an interactive semi-supervised optimizer, for the inner workings of the WHUN algorithm and the SNEAK optimizer are very alike, differing on how to select the next node to be queried and the fact that SNEAK goes to the model to recover the labels for the data when querying a node. The recursive bi-clustering used by both approaches, that made this possible, can be also used as a non-parametric sampling method, allowing for extrapolation between examples in leaves of the recursively clustered data. This is what makes it possible for our methods to set a "Budget" for model evaluations and still achieve good results.

2.4 Multi-objective optimization on expensive evaluation models

One fundamental challenge in many problems in software engineering and other domains is to be able to optimize towards multiple objectives. For a concrete example, one has to balance the number of hours put into tasks, the budget for each task and the quality of each product generated when managing a software development project. Usually there is not a single model that will excel in all objectives, so the best general approach is to identify the best region in space for all objectives and search within those boundaries. This optimal region is named the Pareto frontier. Apart from that, evaluating the objective functions for a model is expensive and noisy. For example, if we take hardware design, the synthesis of a single sample can take hours or even days. The fundamental problem that we will discuss in this section, is how to find this frontier within very few evaluations.

Zuluaga et al. [81] developed PAL, an approach to predict Pareto-optimal points using Gaussian models to predict the objective function after training in a small subset of the original data. With their approach they were able to reach good results after evaluating around 10% of the samples on simple models (< 10 features). However the objective of this work was to classify every data point with a cheaper function, instead of looking for a single optimal solution.

Gelbart et al. [26] developed a Bayesian optimization extension in order to look for a single optimal solution. They formulate the given problem using probabilistic constraints, allowing a user to directly express the trade-off between cost and risk by specifying a confidence parameter. They are able to reach good results within very few evaluations. However, their approach was only tested for very small problems (in terms of number of variables), and is not exact, in the sense that for the problems treated throughout this paper, there is a high likelihood that using these methods would result in very few actual valid solutions at the end of the process. Which was shown by Lustosa et al. [36] about Araujo et al. [6].

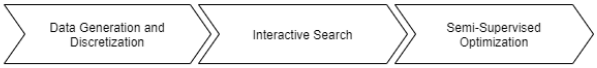

Figure 2: General SNEAK workflow

Nair et al. [50] developed FLASH, a sequential model-based method (SMBO), as a way to quickly assess and find optimal configurations for software systems. This approach similarly to Gelbart et al. is also generative, being non-exact. However for the set of problems towards which this approach is applicable, FLASH is capable of finding near optimal points within very few evaluations (2 to 4% of the configuration space). While this is extremely efficient for the problems it solves, when we are talking about super complex models with 2^{128} possibilities, 2 to 4% of this space is still an impossibly high number.

Chen et al. [12] solved this problem by using oversampling techniques. Their methods are much more suitable for the problems we deal with here. Using a recursive clustering method, the SWAY algorithm is capable of finding results within very few evaluations ($O(log(N))$ where N is the number of samples generated). Their results are two orders of magnitude faster than other state-of-the-art Multi objective optimizers, while achieving similar results.

SNEAK uses a similar approach to Chen et al's oversampling methods in order to perform the final evaluations in the post-processing step of our architecture. Given that the interactive search part of our methods is capable of reducing the original space to at most $\sqrt{N}$ candidate solutions. The application of these oversampling methods further reduce the space to at most $\sqrt[4]{n}$, meaning that SNEAK will only evaluate the latter. Also the knowledge of the existence of a representative sample for this surviving set through these techniques allows us to limit further evaluations according to a fixed budget while maintaining good results.

3 THE METHOD

3.1 Pre-processing - Discretizing

As shown in Figure 2 SNEAK's workflow starts by generating a dataset of valid solutions for a model, either by utilizing a SAT Solver for CNF Models, or other generative methods for different models.

Once this dataset is generated, we discretize any continuous features. Any other discretization method could be applied here as a pre-processor, however we opted for a naive approach of dividing any continuous variable into 4 bins, splitting the data while minimizing entropy and then recursing once more on each half. This method then creates 4 binary features for each original continuous feature.

The reasoning behind this pre-processing step is to expand the methods proposed by Lustosa et al [36]. Their methods while very efficient when compared to other methods in the literature, lacked generality beyond CNF-based models. Using a discretization pre-processor on continuous variables within the distribution of the dataset allows for the use of WHUN [36] towards broader datasets.

3.2 Interactive Search

At the core of SNEAK is a slightly modified version of the WHUN [36] algorithm. WHUN can be described as an interactive search process over a recursive cluster structure. It initially recursively divides the data up to $\sqrt{n}$ sized bins. This is done via Chen et al.'s boolean radial distance metric [12].

From there WHUN ranks all nodes of the tree-like structure to decide where to start exploring the space. These nodes are scored using four heuristics:

- *Refraction*, which can be summarized as never ask the same thing twice;
- *Entropy*, measuring the variability of each variable of the dataset;
- *Depth of first difference*, a measure of how early in the tree-like structure a feature differs in the sub-trees;
- *Differences*, how many different variables are there between the representatives of each side of the current node.

Refraction is a value assigned to each feature and each node. Initially it is 0, but once we query the human on a node or feature it's asked value becomes 1. *Entropy* is calculated at each point of the loop for the remaining candidate solutions. The surviving data at each loop will have solutions that mention different features $F = f_1, f_2, \cdot$, and each of those features will have *true* or *false* frequencies defined by n and $N - n$. The entropy is then calculated as the effort to recreate that distribution:

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

The feature with highest entropy is the one that will exclude the most number of remaining solutions. *Depth of first difference* is a measure of how early in the tree structure a certain feature differs in the representatives of the sub-tress in the subset of N remaining candidate solutions. At any point of the loop, the depth is a normalized count of the first difference of that given feature on the tree. Similarly to entropy this feature also selects for nodes that will potentially exclude the largest number of remaining data. Finally *Differences* is a simple xor operation between the representatives of each side of the node. This difference vector is then summed and normalized. As the algorithm progresses, the difference factor becomes smaller and smaller, because the algorithm will be working with solutions of increasingly similarity. The features are ranked in terms of their importance, and so are the nodes using the aforementioned heuristics.

All the heuristics can be summarized by: "*never ask the same thing twice, ask things that provide the largest average cut in the dataset, update your metrics as you go*". After ranking the nodes and selecting the best node WHUN will select up to the top 6 features from either side to query the user on their preference. Once an answer is given, WHUN proceeds to remove the non-selected half from consideration, and also any sample from the selected half that does not contain the selected attributes. This process is repeated until the algorithm runs out of interesting nodes or features to ask, which is to say the metrics described above can no longer differentiate surviving samples.

Originally WHUN would then evaluate all surviving samples, around 15% of the data on average, as seen in Table 2. Given that as discussed previously, in many real life scenarios the evaluation of a solution in a model is an expensive task SNEAK skips this step, substituting it with the process described in the following section.

3.2.1 Domination. When dealing with a single-objective optimization problem, a simple sorting function can rank goal values between candidate solutions. However, when dealing with multi-objective reasoning, those candidates must be ranked across many goal values.

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 [74].

On the WHUN paper by Lustosa et al. [36] a more simplistic approach towards the multi-objective optimization was used by pure weighted aggregation of the dependent variables via the proposed minimization function, however Sayyad et al. [61] have recommended that the Zitler's *continuous domination predicate* [80] should be used in such cases.

Therefore this paper utilizes Zitler's predicate to rank the candidate solutions in order to evaluate the effectiveness of SNEAK.

3.3 Post-Processing - Semi-Supervised Optimization

Once WHUN is stopped before evaluating the candidate solutions, we are left with around 15% of the original data. Given that evaluating this data is expensive we have developed a way to integrate the approach of Chen et al. [12] on their SWAY algorithm to perform a task of semi-supervised optimization on the surviving data. SNEAK uses the FASTMAP [21] random projection algorithm to recursively bi-cluster the data. SNEAK finds two distance points, labeled *east* (E) and *west* (W). The rest of the data is then divided into two, depending on whether or not each sample is closer to E or W. The algorithm then will compare the median point from each cluster, selecting the side that best optimizes towards the goals [80]. Pruning the worst half of the data and recursing on the rest. This is repeated up to clusters of size $\sqrt{N}$ where N is the size of the dataset passed into the SNEAK semi-supervised optimizer

Given a model evaluation budget (B) we apply the above mentioned techniques in order to sample the dataset down to around 0.35% of the original dataset as seen in Table 2.This step after WHUN has a fixed cost of evaluating around 0.08% of the original data. From there we have the option of evaluating all of the surviving data, evaluating a fixed budget of extra samples from the surviving data or even using the extrapolation techniques discussed previously, select the median sample from the surviving distribution as our solution.

4 CASE STUDIES

To explore the effectiveness of this *semi-supervised* approach towards iSBSE, we have selected different models within the three categories stated by Chen et al. [12] as representative of variable levels of difficulty in optimization.

- Model size (number of features)
- Conflicts (number of constraints)
- Decision space (discrete or continuous)

From the literature in both SBSE and iSBSE, most problems will fall under small models, with **no** conflicts on continuous spaces. However, the hardest problems to solve, and the ones used to stress test multi-objective optimization algorithms, are large models with many conflicts on discrete decision spaces.

The models offered here fall within those categories.

4.1 XOMO

XOMO, introduced by Menzies et al. [42], is a general framework that uses Monte Carlo simulations. This model 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). That being said, the three XOMO models offered ,OSP2, GROUND and FLIGHT (ordered from most to least complex), fall within the simplest models according to Chen et al. XOMO possesses a limited number of features, no conflicts and exists in a continuous space.

4.2 POM3

POM3 is a model for exploring the management process of agile development [8, 9, 54]. The objective of POM3 is find an effective configuration nine continuous variables in order to increase completion rates (measure of project termination), reduce idle rates (measure of time efficiency for team members), and decrease the overall cost. Given the nature of how POM3 models all of the requirements of a project, with costs and a prioritization value, along with a list of dependencies. There are conflicts and constraints in the model, although to a lesser degree than one of the later discussed models. The complexity of our POM3 models follows what was stated by Chen et al. [12], with POM3a being the least complex and POM3c being the most complex of them all. These are considered intermediate complexity models due to them having a small number of features and existing in a continuous space.

4.3 Software Product Lines

A Software Product Line (SPL) is a collection of possible software products, defined by a model. These will share some core functionality. Rocha and Fantinato [18] remind us that product lines are a generalization of the SPL architecture design for business process management. This representation defines the multiple legal variations of the core functionality. For example, a phone can be described as an SPL, the core functionality being making and receiving calls, we can look to modern smartphones and all of the different configurations, camera, screens, battery, sensors, processors, etc. These are all possible products of an SPL. In this paper we have used the two SPL models provided by Lustosa et al. [36] SCRUM is a model defining the possible legal configurations of the SCRUM organization framework. The SCRUM model contains 128 features and more than 200 constraints. These constraints are so complicated that out of the 2^{128} possible solutions less than 1% is valid. Billing is a simpler model that defines valid configurations for a billing software system. Containing 88 features and 166 constraints, while not as complex as the SCRUM model, the Billing model is complex enough to where less than 1% of its configuration space is valid. Both these models deal with binary variables, each representing a possible feature of the final product. In both models, we seek to minimize costs, efforts and predicted defects, while maximizing features associated with previous success. These are considered as models of high complexity due to the large amount of variables and constraints, and the fact that it's features are discrete.

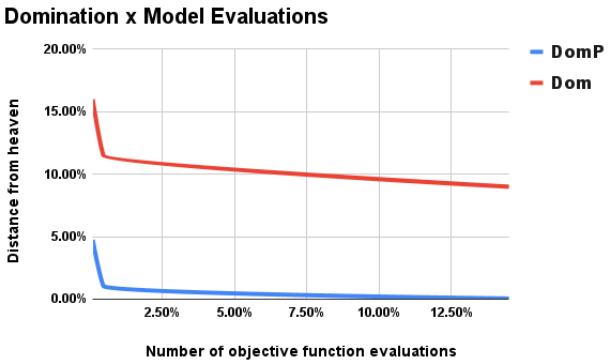

Figure 3: Performance increase with evaluation budget.

4.4 The experiment

For our experiment we have defined three procedures:

- WHUN [36] with the pre-processing step for the XOMO and POM3 models
- SNEAK with an unlimited evaluation budget
- SNEAK with a minimal evaluation budget (SNEAK+)

We have then generated 20 thousand valid samples from each of the models described in the previous sections, through different methods:

- SCRUM & Billing: Generated with a SAT Solver
- XOMO, POM3: Generated utilizing the methods provided by Chen et al. [12]

We then executed the pre-processing on the XOMO and POM3 models to obtain a discretized dataset of the same 20 thousand samples. This increased by a factor of ≈ 4 the number of variables in the XOMO and POM3 datasets. From here, we have executed each procedure on each dataset 20 times. Recording the number of evaluations, CPU-time and their associated results.

5 RESULTS

5.1 Research Questions

RQ1: Does discretizing the continuous variables impact on the performance of the algorithm?

When running both pure WHUN and SNEAK on datasets with discretized variables (Flight, Ground, OSP2 and the POM3x), there seems to be some small degradation on the quality of the results as seen on Table 2. Even so, the results are comparable in performance with the non discretized datasets, not only that but as seen on Table 3 the performance of the algorithm is still very high on discretized datasets given that sensible preferences are applied towards the

Table 2: Performance Comparison on 8 Datasets (lower is better). Dom refers to the distance to heaven according to Zistler's predicate. DomP refers to the distance to heaven according to Zistler's predicate, taking into account human preference as a new goal.

	WHUN			SNEAK		
Dataset	**Dom**	**DomP**	**% Dataset Evaluated**	**Dom**	**DomP**	**% Dataset Evaluated**
SCRUM	8.00%	0.01%	14.00%	11.00%	0.25%	0.50%
Billing	7.00%	0.01%	10.50%	9.00%	0.28%	0.50%
Flight	10.00%	0.05%	20.00%	10.00%	0.95%	0.45%
Ground	12.00%	0.05%	22.00%	15.00%	1.20%	0.45%
OSP2	7.00%	0.05%	19.00%	12.00%	1.50%	0.45%
POM3a	8.00%	0.05%	15.00%	13.00%	2.00%	0.30%
POM3b	11.00%	0.05%	12.00%	12.00%	1.00%	0.40%
POM3c	10.00%	0.05%	14.00%	11.00%	1.10%	0.45%

	SNEAK+		
Dataset	**Dom**	**DomP**	**% Dataset Evaluated**
SCRUM	15.00%	2.00%	0.08%
Billing	18.00%	3.00%	0.05%
Flight	15.00%	3.00%	0.08%
Ground	16.00%	6.00%	0.08%
OSP2	17.00%	6.50%	0.08%
POM3a	17.00%	7.00%	0.05%
POM3b	15.00%	5.00%	0.08%
POM3c	16.00%	4.50%	0.08%

search process. Even so the results support that on average discretized datasets will return results from 1.5% to 6% of the best results depending on the allowed budget for evaluations.

RQ2: What is the impact of the pre and post processing steps to the runtime and scalability of the algorithm?

The pre-processing step was not included into the SNEAK algorithm time calculation, being it a requirement for the algorithm to operate. This step can be done only once and a dataset generated from it can be reused multiple times. Nonetheless a simple entropy based binary discretizer was able to run in all versions of the XOMO and POM datasets in seconds.

However when we analyzed the cost of evaluation in the original WHUN algorithm, we noticed that around 80 to 90% of the CPU time was used towards evaluation of the surviving solutions. On average WHUN would run in 0.5s and evaluate solutions for the remaining of the time in all cases covered by Lustosa et al. [36]. This was not as impactful on their paper given that the models they were using had very inexpensive evaluation functions. Given this fact, even with the overhead of the secondary semi supervised algorithm, SNEAK runs 85 to 80% faster depending on the evaluation budget when compared to WHUN.

Exploring further we did a trial run on the 1000 variables model provided by Lustosa et al. [36]. In the case of the WHUN algorithm, the evaluations process takes up about 97% of the CPU time, totaling on average 14.5s of runtime. SNEAK was able to run to completion on this algorithm in 2 to 2.5s on average depending on the evaluation budget.

In one other further trial we have simulated an expensive function for the model by adding a one second sleep in the processing for each evaluation. For the SCRUM dataset, with this simulation WHUN ran in 25 minutes (≈1500 evaluations). Meanwhile SNEAK finished running in about 51 seconds (≈ 50 evaluations).

So the post processing step is very important in guaranteeing the scalability of the algorithm when facing either large datasets or time consuming evaluation functions.

RQ3: What are the trade-offs when we reduce the number of evaluations?

By definition, the best possible strategy to select the best solution in a list of solutions would be to evaluate them all, sort them and select the best solution. WHUN [36] does exactly that. As discussed previously this step is extremely CPU intensive, even when dealing with low cost evaluation functions it can take over of about 80% of the CPU time. SNEAK's strategy, while sub optimal when compared to WHUN, still yields good results. As seen in Table 2 SNEAK regardless of budget strategy is still able to reach scores within the top 5% of all available solutions, and given a small budget of evaluations we can consistently reach scores within the top 1% of all available solutions.

When we put our experiments into perspective we obtain the chart in Figure 3, where we can clearly see a hockey stick like pattern emerge in both dominations (with or without human preference). This pattern indicates that while we are able to sharply improve results within few evaluations, after some point the curve starts to stabilize, with reducing gains in performance for similar amounts of effort. That as a given, we are able to establish a low number of evaluations being necessary for finding solutions within the top 1%. This also establishes an expectation on the quality of the solution given the budget of evaluations for the problem at hand.

5.2 Using an automated oracle

Given the nature of the oracle used in this paper in lieu of a human participant, it is expected that random selections of model characteristics are not optimal to search. In general it is expected that expert knowledge will function as a strong guide towards a good region in space from which we can select candidate solutions.

In Table 3 we present one of the cases when the decision made by the Oracle guided us to a much more optimal space. Allowing us on all treatments to reach results within 4% of the best available solutions in the dataset, and within 2% when taking preference into consideration. Studying the decisions made in this specific run, we found that the oracle had selected characteristics associated with a good software cost estimation model for XOMO. Which in turn better guided SNEAK towards better results.

Table 3: One of the best runs (for expected values seen in 20 repeats, see Table 2

Sample Run (Flight Dataset)			
Treatment	WHUN	SNEAK	SNEAK+
Dominance	0.80%	1.50%	3.60%
Dominance + Preference	0.30%	0.89%	1.51%
% Dataset Evaluated	15.5%	0.45%	0.08%

6 DISCUSSION: WHY TO SNEAK

Here we discuss the advantages of SNEAKing and its uniqueness over other methods in iSBSE.

Generality: The usage of a binary discretizer as a pre-processing step makes it so that SNEAK is capable of dealing with any configuration problem and any other problem that can be reduced somehow to a binary decision space. Thus pratically any model can be reduced to a form acceptable by SNEAK's architecture. The fact that a dataset of valid solutions is pre-generated, and then sampled from, also eliminates any problems that could arise from constraints when generating new solutions through non exact algorithms.

Scalability: WHUN [36] on it's own had solved scalability in regards to the number of variables and constraints in a model. Other works in iSBSE tend to be limited in this, either because of CPU-time constraints, or due to the excessive cognitive fatigue generated by their methods. However, WHUN did not address the problems arising from models with extremely expensive evaluations of their objective function [81]. SNEAK via the usage of semi-supervised learning techniques combined with multi-objective optimization techniques is capable of reducing by a large margin the total number of evaluations necessary to reach a *"good enough"* solution.

Exactness: Before SNEAK , and to the best of our knowledge, only two other methods in the iSBSE literature have been used exact techniques to solve their set of problems (Palma et al. [52] and Lustosa et al. [36]). Exact algorithms are extremely important on problems containing a large quantity of constraints. SNEAK is an improvement over MAX-SMT [52] and WHUN [36] on it's generality and scalability.

Versatility: SNEAK is not limited to a small subset of problems. In fact any problem that can be reduced to it's architectural requirements can possibly be solved by SNEAK. Given the decoupling of the components in the architecture SNEAK could be modified to better fit towards different problems.

7 THREATS TO VALIDITY

Given this to be an empirical study, biases can affect the final results. Therefore, the conclusions made from this work need to be considered with the following issues in mind:

Evaluation Bias As discussed in subsection 5.2, we have utilized an automated oracle instead of human, this was done for the reasons discussed in Lustosa et al's [36] work. Using humans to test this algorithm for the number of runs would be incredibly expensive. The experiments detailed here were run for over 500 times, and using the rationale of Valerdi [73]. Although a single run is relatively inexpensive for humans due to the low number of evaluations and their size, 500 runs of the algorithm would mean the evaluation of over 8000 variables in over 2500 interactions. If we take Valerdi's numbers as a fact for this case it would result in over 2000 man hours of evaluations.

Sampling Bias This will always threathen an empirical study, we have followed advice from previous work in the literature [12] in selecting and defining our datasets in order to cover the spectrum of possible data. However, the behavior of SNEAK should still be evaluated towards much larger models, and models defining different problems in Software Engineering.

Algorithm bias For our pre-processing step we have used a naive approach to discretize continuous variables in bins of equal entropy. Other techniques may exist that will be more effective and better represent continuous variables as binary values. Also the WHUN algorithm [36] might not be the most effective solution for models such as XOMO where there are no constraints. Still the architecture defined by SNEAK can be considered generic enough for a large variety of SE and non SE related problems.

8 CONCLUSION

When AI tools are capable of finding a large amount of solutions, we need iSBSE to help us sort out which solution applies towards our preferences. However, current iSBSE methods lack one or more of three factors: Generality, Versatility and Scalability.

Here we extend the work of Lustosa et al. [36] by offering an iSBSE general purpose multi-objective optimizer that uses the WHUN algorithm at it's core, SNEAK. Using discretization techniques paired with semi-supervised learning we are able to dramatically increase the scope of problems treated by this algorithm, while also creating a possibility for a much reduced number of model evaluations within a fixed budget. The usage of these techniques, to the best of our knowledge had not yet been applied in iSBSE literature.

Compared to the previous state-of-the-art [36], SNEAK is capable of solving a much broader range of issues by simply including it's pre-processing step. It also cuts down on the number of objective function evaluations by a factor of ≈ 40 with maximal budget and of ≈ 200 with minimal budget. Even with such dramatic cuts in

evaluations SNEAK is still capable of finding solutions within the top 1% of the best solutions in the dataset.

Therefore we recommend the use of *semi-supervised* techniques in conjunction with *existing iSBSE methods* in order to dramatically decrease the number of model evaluations required to reach a "good enough" solution. We also reiterate the advice made by Lustosa et al. [36]. For better iSBSE, it is better to use data-mining techniques instead of patching an existing search-based algorithm to focus the subsequent search within the set of learned human preferences.

ACKNOWLEDGMENTS

.....

REFERENCES

[1] Raja Ben Abdessalem, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. 2018. Testing Vision-based Control Systems Using Learnable Evolutionary Algorithms. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. ACM, New York, NY, USA, 1016–1026. https://doi.org/10.1145/3180155.3180160

[2] Amritanshu Agrawal, Wei Fu, and Tim Menzies. 2018. What is wrong with topic modeling? And how to fix it using search-based software engineering. *Information and Software Technology* 98 (2018), 74–88.

[3] Amritanshu Agrawal and Tim Menzies. 2018. 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, 1050–1061.

[4] Mohammed Hasan Ali, Bahaa Abbas Dawood Al Mohammed, Alyani Ismail, and Mohamad Fadli Zolkipli. 2018. A new intrusion detection system based on Fast Learning Network and Particle swarm optimization. *IEEE Access* 6 (2018), 20255–20261.

[5] Boukhdhir Amal, Marouane Kessentini, Slim Bechikh, Josselin Dea, and Lamjed Ben Said. 2014. 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, 31–45.

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

[7] Gabriele Bavota, Filomena Carnevale, Andrea De Lucia, Massimiliano Di Penta, and Rocco Oliveto. 2012. Putting the developer in-the-loop: an interactive GA for software re-modularization. In *International Symposium on Search Based Software Engineering*. Springer, 75–89.

[8] Barry Boehm and Richard Turner. 2003. *Balancing agility and discipline: A guide for the perplexed*. Addison-Wesley Professional.

[9] Barry Boehm and Richard Turner. 2003. Using risk to balance agile and plan-driven methods. *Computer* 36, 6 (2003), 57–66.

[10] Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien (Eds.). 2006. *Semi-Supervised Learning*. The MIT Press. http://dblp.uni-trier.de/db/books/collections/CSZ2006.html

[11] Jianfeng Chen, Vivek Nair, Rahul Krishna, and Tim Menzies. 2018. "Sampling" as a Baseline Optimizer for Search-based Software Engineering. *IEEE Transactions on Software Engineering* (2018).

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

[13] Jianfeng Chen, Vivek Nair, and Tim Menzies. 2018. Beyond evolutionary algorithms for search-based software engineering. *Information and Software Technology* 95 (2018), 281–294.

[14] Nan-Hsing Chiu and Sun-Jen Huang. 2007. The adjusted analogy-based software effort estimation based on similarity distances. *Journal of Systems and Software* 80, 4 (2007), 628–640.

[15] Altino Dantas, Italo Yeltsin, Allysson Allex Araújo, and Jerffeson Souza. 2015. Interactive software release planning with preferences base. In *International Symposium on Search Based Software Engineering*. Springer, 341–346.

[16] Andre B De Carvalho, Aurora Pozo, and Silvia Regina Vergilio. 2010. A symbolic fault-prediction model based on multiobjective particle swarm optimization. *Journal of Systems and Software* 83, 5 (2010), 868–882.

[17] Thiago do Nascimento Ferreira, Allysson Allex Araújo, Altino Dantas Basílio Neto, and Jerffeson Teixeira de Souza. 2016. Incorporating user preferences in ant colony optimization for the next release problem. *Applied Soft Computing* 49 (2016), 1283–1296.

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

[19] Xin Du, Xin Yao, Youcong Ni, Leandro L Minku, Peng Ye, and Ruliang Xiao. 2015. An evolutionary algorithm for performance optimization at software architecture level. In *Evolutionary Computation (CEC), 2015 IEEE Congress on*. IEEE, 2129–2136.

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

[21] Christos Faloutsos and King-Ip Lin. 1995. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*. 163–174.

[22] Martin S Feather and Tim Menzies. 2002. Converging on the optimal attainment of requirements. In *Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on*. IEEE, 263–270.

[23] Wei Fu and Tim Menzies. 2017. Easy over hard: A case study on deep learning. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, 49–60.

[24] Wei Fu, Tim Menzies, and Xipeng Shen. 2016. Tuning for software analytics: Is it really necessary? *Information and Software Technology* 76 (2016), 135–146.

[25] Wei Fu, Tim Menzies, and Xipeng Shen. 2016. Tuning for software analytics: Is it really necessary? *Information and Software Technology* 76 (2016), 135–146.

[26] Michael A Gelbart, Jasper Snoek, and Ryan P Adams. 2014. Bayesian optimization with unknown constraints. *arXiv preprint arXiv:1403.5607* (2014).

[27] Adnane Ghannem, Ghizlane El Boussaidi, and Marouane Kessentini. 2013. Model refactoring using interactive genetic algorithm. In *International Symposium on Search Based Software Engineering*. Springer, 96–110.

[28] Mark Harman, S Afshin Mansouri, and Yuanyuan Zhang. 2012. Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys (CSUR)* 45, 1 (2012), 1–61.

[29] Sun-Jen Huang and Nan-Hsing Chiu. 2006. Optimization of analogy weights by genetic algorithm for software effort estimation. *Information and software technology* 48, 11 (2006), 1034–1045.

[30] Sun-Jen Huang, Nan-Hsing Chiu, and Li-Wei Chen. 2008. Integration of the grey relational analysis with genetic algorithm for software effort estimation. *European Journal of Operational Research* 188, 3 (2008), 898–909.

[31] Jonathan Hui. 2019. Machine Learning —Expectation-Maximization Algorithm (EM). https://jonathan-hui.medium.com/machine-learning-expectation-maximization-algorithm-em-2e954cb76959

[32] K Vinay Kumar, Vadlamani Ravi, Mahil Carr, and N Raj Kiran. 2008. Software development cost estimation using wavelet neural networks. *Journal of Systems and Software* 81, 11 (2008), 1853–1867.

[33] Elizaveta Levina and Peter Bickel. 2005. Maximum Likelihood Estimation of Intrinsic Dimension. In *Advances in Neural Information Processing Systems*, L. Saul, Y. Weiss, and L. Bottou (Eds.), Vol. 17. MIT Press. https://proceedings.neurips.cc/paper/2004/file/74934548253bcab8490ebd74afed7031-Paper.pdf

[34] Yun Lin, Xin Peng, Yuanfang Cai, Danny Dig, Diwen Zheng, and Wenyun Zhao. 2016. Interactive and guided architectural refactoring with search-based recommendation. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 535–546.

[35] Yi Liu, Taghi M Khoshgoftaar, and Naeem Seliya. 2010. Evolutionary optimization of software quality modeling with multiple repositories. *IEEE Transactions on Software Engineering* 36, 6 (2010), 852–864.

[36] Andre Lustosa and Tim Menzies. 2021. Preference Discovery in Large Product Lines. *arXiv preprint arXiv:2106.03792* (2021).

[37] Bogdan Marculescu, Robert Feldt, and Richard Torkar. 2013. 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, 102–107.

[38] Bogdan Marculescu, Robert Feldt, Richard Torkar, and Simon Poulding. 2015. An initial industrial evaluation of interactive search-based testing for embedded software. *Applied Soft Computing* 29 (2015), 26–39.

[39] Bogdan Marculescu, Simon Poulding, Robert Feldt, Kai Petersen, and Richard Torkar. 2016. Tester interactivity makes a difference in search-based software testing: A controlled experiment. *Information and Software Technology* 78 (2016), 66–82.

[40] George Mathew, Tim Menzies, Neil A Ernst, and John Klein. 2017. "SHORT" er Reasoning About Larger Requirements Models. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*. IEEE, 154–163.

[41] Tim Menzies, Oussama Elrawas, Jairus Hihn, Martin Feather, Ray Madachy, and Barry Boehm. 2007. The Business Case for Automated Software Engineering. In *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering* (Atlanta, Georgia, USA) *(ASE '07)*. ACM, New York, NY, USA, 303–312. https://doi.org/10.1145/1321631.1321676

[42] Tim Menzies and Julian Richardson. 2005. Xomo: Understanding development options for autonomy. In *COCOMO forum*, Vol. 2005.

[43] Leandro L Minku and Xin Yao. 2013. 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, 8.

[44] Leandro L Minku and Xin Yao. 2013. Software effort estimation as a multiobjective learning problem. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 22, 4 (2013), 35.

[45] Mohamed Wiem Mkaouer. 2016. Interactive code smells detection: An initial investigation. In *International Symposium on Search Based Software Engineering*. Springer, 281–287.

[46] Mohamed Wiem Mkaouer, Marouane Kessentini, Slim Bechikh, Kalyanmoy Deb, and Mel Ó Cinnéide. 2014. Recommendation system for software refactoring using innovization and interactive dynamic optimization. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*. 331–336.

[47] N Monmarché, G Nocent, M Slimane, G Venturini, and P Santini. 1999. 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, 640–645.

[48] Stephen L Morgan and Stanley N Deming. 1974. Simplex optimization of analytical chemical methods. *Analytical chemistry* 46, 9 (1974), 1170–1181.

[49] Vivek Nair, Tim Menzies, Norbert Siegmund, and Sven Apel. 2017. Using bad learners to find good configurations. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. 257–267.

[50] Vivek Nair, Zhe Yu, Tim Menzies, Norbert Siegmund, and Sven Apel. 2018. Finding faster configurations using flash. *IEEE Transactions on Software Engineering* 46, 7 (2018), 794–811.

[51] Adriano LI Oliveira, Petronio L Braga, Ricardo MF Lima, and Márcio L Cornélio. 2010. GA-based method for feature selection and parameters optimization for machine learning regression applied to software effort estimation. *information and Software Technology* 52, 11 (2010), 1155–1166.

[52] Francis Palma, Angelo Susi, and Paolo Tonella. 2011. 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*. 48–58.

[53] Annibale Panichella, Bogdan Dit, Rocco Oliveto, Massimiliano Di Penta, Denys Poshyvanyk, and Andrea De Lucia. 2013. 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, 522–531.

[54] Dan Port, Alexy Olkov, and Tim Menzies. 2008. Using simulation to investigate requirements prioritization strategies. In *2008 23rd IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 268–277.

[55] Juan C Quiroz, Sushil J Louis, Anil Shankar, and Sergiu M Dascalu. 2007. Interactive genetic algorithms for user interface design. In *2007 IEEE congress on evolutionary computation*. IEEE, 1366–1373.

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

[57] Ali Safaa Sadiq, Basem Alkazemi, Seyedali Mirjalili, Noraziah Ahmed, Suleman Khan, Ihsan Ali, Al-Sakib Khan Pathan, and Kayhan Zrar Ghafoor. 2018. An Efficient IDS Using Hybrid Magnetic Swarm Optimization in WANETs. *IEEE Access* 6 (2018), 29041–29053.

[58] Federica Sarro, Sergio Di Martino, Filomena Ferrucci, and Carmine Gravino. 2012. 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, 1215–1220.

[59] Federica Sarro, Filomena Ferrucci, and Carmine Gravino. 2012. Single and Multi Objective Genetic Programming for Software Development Effort Estimation. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing* (Trento, Italy) *(SAC '12)*. ACM, New York, NY, USA, 1221–1226. https://doi.org/10.1145/2245276.2231968

[60] Federica Sarro, Alessio Petrozziello, and Mark Harman. 2016. Multi-objective software effort estimation. In *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*. IEEE, 619–630.

[61] Abdel Salam Sayyad, Tim Menzies, and Hany Ammar. 2013. 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, 492–501.

[62] Christopher L Simons and Ian C Parmee. 2009. An empirical investigation of search-based computational support for conceptual software engineering design. In *2009 IEEE International Conference on Systems, Man and Cybernetics*. IEEE, 2503–2508.

[63] Christopher L Simons and Ian C Parmee. 2012. Elegant object-oriented software design via interactive, evolutionary computation. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42, 6 (2012), 1797–1805.

[64] Christopher L Simons, Ian C Parmee, and Rhys Gwynllyw. 2010. Interactive, evolutionary search in upstream object-oriented class design. *IEEE Transactions on Software Engineering* 36, 6 (2010), 798–816.

[65] Hideyuki Takagi. 1998. Interactive evolutionary computation: System optimization based on human subjective evaluation. In *IEEE International Conference on Intelligent Engineering Systems*, Vol. 1998. 17–19.

[66] Hideyuki Takagi. 2000. Active user intervention in an EC search. In *Proceedings of the Fifth Joint Conference on Information Sciences, JCIS 2000*. 995–998.

[67] Hideyuki Takagi. 2001. Interactive evolutionary computation: Fusion of the capabilities of EC optimization and human evaluation. *Proc. IEEE* 89, 9 (2001), 1275–1296.

[68] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, and Kenichi Matsumoto. 2016. Automated parameter optimization of classification techniques for defect prediction models. In *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*. IEEE, 321–332.

[69] Paolo Tonella, Angelo Susi, and Francis Palma. 2013. Interactive requirements prioritization using a genetic algorithm. *Information and software technology* 55, 1 (2013), 173–187.

[70] Christoph Treude and Markus Wagner. 2019. Predicting Good Configurations for GitHub and Stack Overflow Topic Models. In *Proceedings of the 16th International Conference on Mining Software Repositories* (Montreal, Quebec, Canada) *(MSR '19)*. IEEE Press, Piscataway, NJ, USA, 84–95. https://doi.org/10.1109/MSR.2019.00022

[71] Luigi Troiano, Cosimo Birtolo, and Roberto Armenise. 2016. A validation study regarding a generative approach in choosing appropriate colors for impaired users. *SpringerPlus* 5, 1 (2016), 1–26.

[72] Huy Tu and Tim Menzies. 2021. FRUGAL: Unlocking SSL for Software Analytics. arXiv:2108.09847 [cs.SE]

[73] Ricardo Valerdi. 2010. Heuristics for systems engineering cost estimation. *IEEE Systems Journal* 5, 1 (2010), 91–98.

[74] Tobias Wagner, N. Beume, and B. Naujoks. 2006. Pareto-, Aggregation-, and Indicator-Based Methods in Many-Objective Optimization. In *EMO*.

[75] Jinshui Wang, Xin Peng, Zhenchang Xing, and Wenyun Zhao. 2013. Improving feature location practice with multi-faceted interactive exploration. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 762–771.

[76] Shi Zhong, Taghi M Khoshgoftaar, and Naeem Seliya. 2004. Analyzing software measurement data with clustering techniques. *IEEE Intelligent Systems* 19, 2 (2004), 20–27.

[77] Dengyong Zhou, Olivier Bousquet, Thomas Lal, Jason Weston, and Bernhard Schölkopf. 2004. 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. https://proceedings.neurips.cc/paper/2003/file/87682805257e619d49b8e0dfdc14affa-Paper.pdf

[78] Xiaojin Zhu and Zoubin Ghahramani. 2002. *Learning from Labeled and Unlabeled Data with Label Propagation*. Technical Report.

[79] Xiaojin Jerry Zhu. 2005. Semi-supervised learning literature survey. (2005).

[80] Eckart Zitzler and Simon Künzli. 2004. Indicator-Based Selection in Multiobjective Search. In *PPSN*.

[81] Marcela Zuluaga, Guillaume Sergent, Andreas Krause, and Markus Püschel. 2013. Active learning for multi-objective optimization. In *International Conference on Machine Learning*. PMLR, 462–470.