Active Coevolutionary Learning of Requirements Specifications from Examples

Marcel Wever, Lorijn van Rooijen, Heiko Hamann*
Department of Computer Science, University of Paderborn, Germany
Institute of Computer Engineering, University of Lübeck, Germany

Abstract

Within software engineering, requirements engineering starts from imprecise and vague user requirements descriptions and infers precise, formalized specifications. Techniques, such as interviewing by requirements engineers, are typically applied to identify the user's needs. We want to partially automate even this first step of requirements elicitation by methods of evolutionary computation. The idea is to enable users to specify their desired software by listing examples of behavioral descriptions. Users initially specify two lists of operation sequences, one with desired behaviors and one with forbidden behaviors. Then, we search for the appropriate formal software specification in the form of a deterministic finite automaton. We solve this problem known as grammatical inference with an active coevolutionary approach following Bongard and Lipson (2005). The coevolutionary process alternates between two phases: (A) additional training data is actively proposed by an evolutionary process and the user is interactively asked to label it; (B) appropriate automata are then evolved to solve this extended grammatical inference problem. Our approach leverages multi-objective evolution in both phases and outperforms the state-of-the-art technique (Bongard and Lipson, 2005) for input alphabet sizes of three and more, which are relevant to our problem domain of requirements specification.

1 Introduction and Related Work

We take an active, multi-objective, coevolutionary approach to synthesizing requirements specifications from examples. The specification of requirements on a software system forms a vital part of the development process of the system. Traditionally, requirements engineers support users in the elicitation of their requirements. The requirements formulated by users are likely to be incomplete and imprecise. As the requirements specification should form the foundation for the subsequent software development process, these vague user

*published at GECCO 2017, http://dx.doi.org/10.1145/3071178.3071258

expectations should be transformed into a more formal requirements specification of the desired system. We use search-based software engineering based on a genetic algorithm to develop a semi-automatic approach for formalizing requirements specifications. Our vision is to use such approaches in future dynamic software service markets where software services are searched and composed (semi-)automatically on a large scale.

1.1 Search-Based Software Engineering

In search-based software engineering, problems from software engineering are reformulated as search problems, on which metaheuristic search techniques may be applied (Harman et al., 2012). Among the topics studied are the problems of constructing various software engineering artifacts (semi-)automatically from examples. These artifacts are traditionally constructed by human experts. Automatic synthesis would empower users to construct these artifacts independently.

Synthesizing model transformations from example transformations of model instances has been studied in, for example, (Kappel et al., 2012; Kessentini et al., 2012; Faunes et al., 2013; Kühne et al., 2016). Evolutionary approaches as taken, for example, in Faunes et al. (2013) and (Kühne et al., 2016), are promising techniques in this field, since the search space of such *by-example* problems is usually complex. Furthermore, being so-called black-box optimizers, the synthesis problems do not have to be fully understood to obtain good results.

1.2 Dynamic Software Service Markets

While being necessary in a traditional software development setting, formal requirements specifications are indispensable in the setting of dynamic software service markets of the future. We study such markets in the Collaborative Research Center "On-The-Fly Computing"[1]. We envision that a specification is used to automatically search the market for a service fulfilling the requirements. If such a service does not yet exist, the specification is analyzed and decomposed, after which may be searched for appropriate existing services as building blocks. These building blocks may vary from single operations to more complex services. These services are then composed automatically, yielding a new service, tailored to the requirements of the user. Since these steps are guided directly by the requirements specification and should all be performed automatically, a sound formal requirements specification is needed.

However, in the setting of a dynamic software service market, one can typically not rely on a requirements engineer to create such a formal specification. Rather on the contrary, users requesting a service make such a request independently and expect a fast result. This means that users should be empowered to rapidly create a formal requirements specification without support of experts. This challenge forms the motivation for our research: how may formal requirements specifications be created semi-automatically, in a user-friendly manner?

In this paper, we focus on users who are able to write down exemplary behavior, in the form of simplified sequence diagrams, of the service they would like to obtain. These simplified sequence diagrams only represent examples of

[1] https://sfb901.uni-paderborn.de/

the interaction between the user and the service, in the most straightforward manner: as sequences of operations (see Table 1 for an example). Since sophisticated features of sequence diagrams are not used in this simplified version, we expect this approach to be appealing to different types of users, ranging from domain experts to possibly even naive users.

The limited amount of user expertise that is needed for our approach distinguishes our work from other approaches to synthesize behavioral descriptions, such as Mäkinen and Systä (2000); Harel et al. (2005); Lambeau et al. (2008).

1.3 Grammatical Inference

Our specification-by-example approach uses evolutionary computation to generate complete behavioral requirements specifications from input examples. A user only has to provide examples of desired and of prohibited behavior of the service. These examples may be described in the form of simplified sequence diagrams, from which we extract sequences of operations. Our goal is to generalize these examples in such a way, that the resulting specification completely determines the behavior of a service, and this behavior should correspond to the model that the user had in mind. Here, we assume that finite automata are the desired formal behavioral specifications. Hence, creating requirements specifications becomes a search problem in the space of deterministic finite automata (DFA).

This allows us to exploit techniques from the field of grammatical inference, where the aim is to learn a representation of a (typically regular) language from a set of words, each word labeled as being a member or not being a member of the language. Theoretical foundations of grammatical inference were initiated by the work of Gold Gold (1967) and, in an active learning setting, by the work of Angluin (Angluin, 1987). The surveys de la Higuera (2005) and de la Higuera (2010) provide an overview of the various approaches to the many different variants of this problem. It should be noted that the size of the alphabet of the automata is usually taken to be 2 in the literature. In our setting, the alphabet is formed by the set of operation names used in the description of the example behavior of the desired service and could therefore be much larger.

Among the most popular deterministic approaches to grammatical inference is the so-called Evidence Driven State Merging (EDSM) algorithm, first introduced in Lang et al. (1998). The papers Lucas and Reynolds (2003, 2005); Bongard and Lipson (2005); Gómez (2006), on the other hand, take an evolutionary approach to learning deterministic finite automata. In Tsarev and Egorov (2011), finite state input-output machines are evolved. In Lucas and Reynolds (2003), it was found that evolutionary methods can outperform EDSM for target DFAs consisting of less than 32 states.

Our problem of synthesizing requirements specifications basically boils down to the problem studied in grammatical inference (viz. learning an automaton from a set of input examples). The techniques developed in this field, however, do not suffice for the challenges posed by our specific setting. In conventional grammatical inference, the input examples are generated from the target automaton that is to be learned. In our setting, the role of this oracle is played by the user wanting to request a service.

1.4 Active Learning, Coevolution, and Multi-Objective Optimization

As we have to rely on the user to provide us with input examples, we face two challenges. First, in order not to overcharge the user, we can only expect a limited amount of input examples. Second, we do not search for one clearly defined target DFA, but we look for several appropriate automata, from which the user may select the one fitting his expectations best.

Because of the first challenge, our approach focuses on obtaining the *right* input data. To this end, we use active learning: our algorithm actively searches for the most informative data during the evolution, and asks the user to label this data. In such an interactive evolutionary approach, the user interaction should be as efficient as possible, i.e., the queries to the user should be minimized. Following the approaches of (Bongard and Lipson, 2005; Ly and Lipson, 2014), we use coevolution to find the potentially most informative next query. One can view this setup as a competition between two evolutionary algorithms. One population consists of DFAs, which formalize the requirements specifications, and the other population consists of candidate words, about which the user may be queried. The evolution operating on DFAs tries to find an accurate and efficient formalization of the growing input data. The evolution operating on potential queries aims to give rise to that additional input example that creates the biggest disagreement among the currently good DFAs. The fitness of these queries is based on the amount of disagreement they cause among good candidate DFAs, while fitness of the DFAs is based on their accuracy.

Even with the most informative input data, there are many DFAs upto a given size that behave precisely the same with respect to the input data, if the number of input examples is small. This means that there would be large plateaus in the fitness landscape. To overcome this problem, and at the same time deal with the second challenge, we evolve the population of DFAs according to multiple objectives. This is different from the evolutionary approaches mentioned above. Here, we use NSGA-II Deb et al. (2002), which gives as output a set of solutions that are Pareto-optimal with respect to the different objectives. Using multi-objective evolution allows us to focus on different quality aspects of the candidates.

We do not only use multi-objective evolution in the DFA population, but also in the population of potential queries. It turns out that taking the length of a query into consideration as well leads to better performance than focusing solely on the disagreement caused by a query. An additional advantage of using multi-objective optimization in both populations is that this approach may easily be extended with additional objectives in the future.

The evaluation of our approach is given in Sec. 3. In this section, we compare the performance of our approach to that of the approach of Bongard and Lipson (Bongard and Lipson, 2005), for different combinations of alphabet size and automaton size. Our evaluation results show that our approach performs particularly well for larger problem sizes, which correspond to settings relevant for our application of requirements specification-by-example.

2 Approach

Initially, the user provides two sets of simplified sequence diagrams, modeling desired behavior in the one set and prohibited behavior in the other one. Simplified sequence diagrams are trimmed UML sequence diagrams including only operation call messages. Furthermore, the number of involved objects is limited to two: the user and the desired service. The user may use arbitrary operation names and provide as many examples as favored. In order to generalize the provided examples to a protocol describing the behavior of the desired service, we extract all distinct operation names from the two sets of sequence diagrams to form an alphabet Σ. Since the sequence diagrams describe interactions between just two objects and the direction of communication may be encoded in the operation names, these sequence diagrams may also be described as sequences of operation names. These sequences of operation names are words over the extracted alphabet Σ. The training data S for our grammatical inference problem consists of pairs (w, l) of such words w and their respective labels l, which classify a word as either desired or prohibited behavior. From this data, we derive a protocol describing the behavior of an application. The protocol is modeled by a DFA, as defined in the following.

2.1 Genetic Representation of DFAs

A *deterministic finite automaton* (DFA) over an alphabet Σ can be defined by a tuple $A = (\Sigma, Q, \delta, q_o, F)$, where Q is a finite set of states, $\delta : Q \times \Sigma \to Q$ the transition function, $q_0 \in Q$ the initial state and $F \subseteq Q$ a set of accepting states. A word $w = i_1 \dots i_n$ of length n is a sequence of n *input symbols* $i_j \in \Sigma$, and is *accepted* by A, if there exists a sequence $q_0 q_1 \dots q_n$ of states such that $\delta(q_{j-1}, i_j) = q_j$ for $j \in \{1, \dots, n\}$ and $q_n \in F$. Otherwise, it is *rejected*. If δ is a total function, then A is called a *complete* DFA. The *size* of a DFA A is defined by $|Q|$. In the following, we only consider complete DFAs as we can simply represent them as matrices over the set of states. Note that this does not constrain our approach since every DFA can be extended to a complete DFA accepting the same language.

Each sequence diagram is interpreted as an input example (w, l), consisting of a word w and a label l, where l is set to *accept* if w represents desired behavior and to *reject* otherwise. As mentioned before, each operation name is considered as an input symbol, and Σ is the set of all distinct operation names. Altogether, the input examples provided by the user form a set of training data S.

In our approach, each DFA $A = (\Sigma, Q, \delta, q_0, F)$ is represented by two parts. The first part represents (Σ, Q, δ, q_0) and consists of a $|Q| \times |\Sigma|$ matrix T over Q. Without loss of generality, we assume Q to be of the form $\{0, \dots, |Q| - 1\}$, $q_0 = 0$, and $\Sigma = \{0, \dots, |\Sigma| - 1\}$. The matrix coordinates of T start from 0, i.e., the top left element is denoted by $T_{0,0}$. Each entry $T_{i,j}$ of the matrix T is assigned the value $\delta(q_i, a_j) =: T_{i,j}$ with $q_i, T_{i,j} \in Q$ and $a_j \in \Sigma$. That is, $T_{i,j}$ contains the state reached from state $q_i \in Q$ reading the input symbol $a_j \in \Sigma$. The second part needs to represent the set of accepting states F of A. Naturally, one would evolve a boolean array, where the i-th value of the array is 1 iff q_i is an *accepting* state. Since evolving this array contributes a factor $2^{|Q|}$ to the search space complexity, instead, we only consider those DFAs with F optimal w.r.t. the given input examples. Using the so-called Smart State Labeling (SSL)

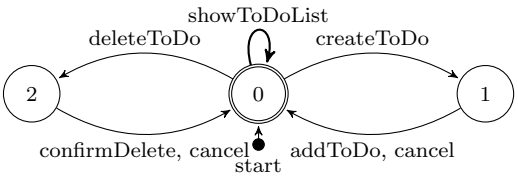


Figure 1: Example DFA of a simple to-do list application.

Desired Behavior	Prohibited Behavior
showToDoList	createToDo confirmDeletion
showToDoList showToDoList showToDoList	createToDo deleteToDo cancel
createToDo addToDo	deleteToDo createToDo
deleteToDo confirmDelete	showToDoList createToDo deleteToDo addToDo cancel
showToDoList deleteToDo confirmDelete showToDoList createToDo addToDo	

Table 1: Example sequences of operation names describing desired and prohibited behavior

algorithm by Lucas and Reynolds (2003, 2005), F is calculated from the given input examples. For each state $q \in Q$, we consider all input examples ending in this state. If the number of input examples labeled with *accept* is greater or equal to the number of the ones labeled with *reject*, state q is considered to be *accepting* (and *rejecting* otherwise). For instance, suppose there are exactly two input examples ending in state q. If at least one of these examples is labeled with *accept*, q is added to F. Otherwise, q is considered to be a *rejecting* state, and therefore, q is not added to F.

Since the second part of the DFA is represented implicitly, a candidate model is completely specified using the matrix T as a genome. This matrix can easily be converted to a sequence of integers, on which standard recombination techniques, such as single-point crossover, may be applied. The overall search space complexity in our setup is $|Q|^{|Q|\cdot|\Sigma|}$. Besides the positive impact of the SSL algorithm on the search space complexity, we are anyhow only interested in finding DFAs with a set of final states that is optimal w.r.t. the input examples obtained from the user. Furthermore, experiments in (Lucas and Reynolds, 2005; Gómez, 2006) have shown that evolutionary algorithms using the SSL algorithm outperform a plain evolutionary algorithm which evolves both transition matrix and set of final states.

2.2 Example

The DFA in Fig. 1 shows an example automaton describing a simple to-do list application. A user of this application may create or delete to-dos. The operation `deleteToDo` triggers a prompt, asking whether the to-do should really be deleted. From this state, the user may either confirm this action or cancel and return to the initial state. When creating a to-do, the user is shown a form for filling in a task description. Then, the user may add the to-do or cancel. The actions `addToDo` and `cancel` let the user return to the initial state.

In order to describe the behavior of the application modeled by the DFA shown in Fig. 1, the user may provide some exemplary sequence diagrams. For instance, these sequence diagrams could be represented by sequences of operation names as shown in Table 1. While the left column describes desired

behavior, prohibited behavior is shown in the right column. As described above, the alphabet Σ is extracted from these examples, as $\Sigma := \{$ `showToDoList`, `deleteToDo`, `createToDo`, `confirmDelete`, `cancel`, `addToDo` $\}$. As explained before, the sequences are interpreted as words over the extracted alphabet Σ and labeled according to the column header.

2.3 Multi-Objective Optimization

Traditional approaches applying evolutionary algorithms to grammatical inference use a single objective (Lucas and Reynolds, 2003, 2005; Bongard and Lipson, 2005; Gómez, 2006). In these approaches, the fitness function for candidate models measures only the proportion of correctly classified input examples. This fitness function can be expressed by

$$f_{\text{all}}(A) = \frac{|\{(w,l) \in S \mid A(w) = l\}|}{|S|}, \tag{1}$$

where A is a finite automaton, S is the set of input examples (training data: w is a word, l its correct label) and $A(w)$ denotes the label returned by the DFA A when reading w. In our scenario, the training data is provided by the user and, therefore, the number of input examples is limited. If the fitness of a DFA would be defined as the accuracy on the small number of provided input examples only, there would be many models with a fitness of 1. To better differentiate between superior and inferior candidate models, a more sophisticated heuristic is required.

For instance, in the case of noisy data with contradiction among the input examples, it might make sense to split the fitness function measuring the accuracy on all input examples into two separate fitness functions. This was proposed in van Rooijen and Hamann (2016), where one fitness function measures the accuracy w.r.t. the positive input examples and another one measures the accuracy w.r.t. the negative examples. In this way, the conflict resolution is delayed to the user, since, for a contradictory input example, the solution could contain one model accepting and one model rejecting the conflicting word.

Here, in order to also take the generalization behavior of a candidate model into account, we use an additional objective to assess the structure of the model. Minimizing the *relevant part* of a candidate model, introduced in (van Rooijen and Hamann, 2016) as

$$f_{\text{rel}}(A) = \frac{\left|\bigcup_{(w,l)\in S}\{q_i \mid q_i \text{ is visited, when } A \text{ reads } w\}\right|}{|Q|}, \tag{2}$$

we force the candidate models to use less states for processing all the training data. As explained in (van Rooijen and Hamann, 2016), using less states indicates more generalization of the examples.

Another advantage of multi-objective optimization is to naturally support preserving a more diverse population of candidate models. We use NSGA-II Deb et al. (2002), maximizing the overall accuracy f_{all} and minimizing the relevant part f_{rel}. NSGA-II uses the concept of Pareto-optimality and sorts individuals by computing Pareto fronts recursively on respective dominated candidate models.

Returning the complete list of Pareto-optimal candidate models as a result to the user, we fully leverage the multi-objective approach. In this way, the user is very flexible in choosing the final model to be used, following his personal preferences of which objectives are most important. As we do not want to overwhelm the user with details about the automata, we only display the options with performance values for each objective and possibly characteristic example words.

We instantiate NSGA-II using a tournament based selection. In a tournament of two candidate models, the individuals are compared using Pareto dominance, following Deb (1998). If using the Pareto dominance relation does not lead to a conclusive result, the two candidate models are compared using the crowding distance, as proposed in Deb et al. (2002). The crowding distance provides a means to estimate the density of other candidate models surrounding the considered candidate model. When comparing two candidate models using the crowding distance in the tournament selection, the candidate model with the larger crowding distance value is preferred.

As genetic operators, we use a single-point crossover and mutation. Recall that we represented DFAs as matrices. We concatenate the rows of a matrix T to a sequence of integers, which is converted to a binary string. On this string, the single-point crossover is applied with a probability of 1. Mutation of an individual is done by flipping a single bit of the binary string. A bit flip is applied to the binary encoding of some integer in T with a probability of $p_M = 1/(|Q| \times |\Sigma|)$. On average, this produces one changed entry in the matrix T belonging to the offspring candidate model.

In our setting, the number of input examples is limited, because it would be impracticable to expect the user to create thousands of sequence diagrams. Thus, many candidate models may be consistent with the given set of input examples. We address this challenge by using multi-objective optimization as a more sophisticated heuristic to deal more efficiently with sparse input data.

An additional challenge is that the input examples are provided by a user who may not be able to estimate the effectiveness of input examples. Therefore, we extend the approach presented so far to an active learning approach, in which interaction with the user allows to refine the provided set of input examples, leading to more accurate candidate models.

2.4 Active Learning

In active learning, the learning algorithm is given access to an oracle that provides additional information about new data points. Hence, the algorithm has the opportunity to strategically augment the given set of input examples. Experiments showed that in the case of grammatical inference, active approaches outperform their passive equivalents, in which instead of strategically deciding on a word, the oracle is queried for the label of a word drawn uniformly at random Bongard and Lipson (2005). These experiments imply that active algorithms need less input examples to infer target models.

Considering our scenario, the user is the only one able to reveal more information about the target model and has to take over the role of the oracle. Hence, after receiving an initial set of input examples provided by the user, the algorithm outputs sequence diagrams as feedback requests to the user. Then the user decides whether the sequence diagram represents desired or prohibited

behavior and labels it accordingly. In this way, less training examples need to be provided by the user in order to obtain good results, and, furthermore, the user is supported by the algorithm in the choice of relevant input examples.

Bongard and Lipson (2005) introduced an active coevolutionary learning algorithm for grammatical inference referred to as the Estimation Exploration Algorithm (EEA). The EEA alternates between two phases until a stopping condition holds. While the estimation phase evolves automata, words are evolved in the exploration phase. Individuals of the estimation phase are referred to as *candidate models* and individuals of the exploration phase as *candidate tests*. First, in the estimation phase, candidate models are evolved in two distinct sub-populations with no interbreeding for g generations. Evolving in two distinct sub-populations maintains more diversity within the overall population. Second, again for g generations, candidate tests are evolved, measuring the fitness by the *disagreement* of some candidate models regarding the label of the word. The disagreement is expressed as

$$f_{\text{dis}}(t) = 1 - 2\left|0.5 - \frac{\sum_{j=1}^{|c|} \text{label}(c_j, t)}{|c|}\right|, \tag{3}$$

where t is a candidate test, c is a set of candidate models and $\texttt{label} : \text{Model} \times \text{Test} \to \{0, 1\}$ is a function, returning 1 if the candidate model accepts the test and 0 otherwise. In the field of active learning, the strategy of letting a set of models vote on a query by disagreement is known as query-by-committee (Settles, 2012). Here, the so-called committee consists of the best individuals of each sub-population evolved in the preceded estimation phase. After the candidate tests have been evolved, the best candidate test is taken as a query to the labeling oracle.

In the case of grammatical inference, the coevolution of candidate models and candidate tests proves beneficial. Adding the evolved candidate test with a maximum disagreement value to the training data, the subsequent evolution of candidate models with this training data results in more sophisticated individuals. For these candidate models, candidate tests are again evolved maximizing the disagreement. Altogether, the coevolution of candidate models and candidate tests leads to an arms race Bongard and Lipson (2005).

Following the idea of EEA, we extend the approach presented in the previous section to an active coevolutionary multi-objective optimization algorithm (MOOA). Analogously, we split the population of candidate models into two and apply NSGA-II to these sub-populations individually.

The algorithm for evolving candidate models outputs a Pareto front of candidate models. The candidate models of the Pareto fronts of each of the sub-populations are used to form the committee in order to calculate the disagreement for the particular candidate tests as described above. Splitting the population ensures that the committee consists of at least two candidate models. In contrast to Bongard and Lipson (2005), we use Pareto fronts rather than best individuals. In this way, the committee may consist of more than two candidate models. This allows for other disagreement values than 0 or 1, causing a gradient in the fitness landscape.

In preliminary experiments, we have found that introducing an additional objective which minimizes the candidate's test length is beneficial, whereas introducing an objective maximizing this length is not. This may be explained

by the fact that errors occurring early on in the automaton lead to erroneous behavior for subsequent actions. Therefore, shorter words with maximum disagreement may have more impact on the quality of the models than longer ones. Note that if a longer word is labeled as prohibited behavior, this might be caused by a single incorrect input symbol. If the word is short, however, it is easier to determine the reason for the label. Therefore, we introduce another objective minimizing the candidate's test length in order to find even more effective words.

2.5 Genetic Representation of Words

For the evolution of words, we again apply NSGA-II. As proposed by Bongard and Lipson (2005), we define the maximum length $\ell_{\max}$ of a word and thus limit the search space complexity to $|\Sigma|^{\ell_{\max}}$. Words are represented as tuples $t = (\ell, w)$, where $w \in \Sigma^{\ell_{\max}}$ and $0 \leq \ell \leq \ell_{\max}$ determines the length of the word: only the first ℓ input symbols of w are considered. A tuple t is represented by an integer string where the first integer denotes the length and the remaining part denotes w. This enables us to use the same genetic operators as for the models. Each entry is modified by mutation with a probability of $1/(\ell_{\max} + 1)$. Additionally, single-point crossover is applied with a probability of 1. However, the actual effectiveness of the crossover is much lower, since only the first ℓ input symbols of a candidate test are considered for fitness evaluation. We apply NSGA-II, aiming for words that cause maximum disagreement and have minimum length. From the Pareto-optimal set of words, we choose the word with the maximum disagreement.

3 Evaluation

The active coevolutionary multi-objective optimization approach proposed in Sec. 2 (MOOA) was compared against the Estimation-Exploration Algorithm (EEA) introduced in Bongard and Lipson (2005). To the best of our knowledge, the approach by Bongard and Lipson (2005) is the state of the art in evolutionary approaches for DFA learning.

3.1 Experiment Setup and Execution

For our scenario, it is crucial to consider alphabets of size greater than two. Hence, for comparing the algorithms EEA and MOOA in settings of different search space complexity, we generate target models differently from Bongard and Lipson (2005). Initially, the target model generation algorithm is given the values $Q = \{0, \ldots, |Q| - 1\}$ and $\Sigma = \{0, \ldots, |\Sigma| - 1\}$. We set $q_0 = 0$, and construct δ drawing states uniformly at random from Q for each combination of states and input symbols. With probability 0.5, a state is added to F. If every state $q \in Q$ is reachable from q_0, the algorithm returns $(\Sigma, Q, \delta, q_0, F)$. Otherwise, we sample a new δ. Note that the resulting target model is not necessarily a minimal DFA. However, $|Q|$ is an upper bound on the number of states necessary to represent the target model. The value $|Q|$ is known to the algorithms. A set of 10 initial input examples is drawn uniformly at random from the set of all words with a maximum length of $\ell_{\max}$.

Queries	States	Alphabet Size	Evolutionary Runs
1,200	4, 8, 32	2	2×50 each combination
1,200	4, 7	7	2×50 each combination
1,200	8	8	2×50
100	3,...,12	3,...,10	2×50 each combination

Table 2: Executed evolutionary runs for evaluation

The conduction of the experiments consists of two parts. First, for the different settings listed in Table 2, target models are generated with the outlined generation routine. Second, the algorithms EEA and MOOA are applied to infer the generated target models, obtaining the initial set of input examples and the value of $|Q|$. Both the population of candidate models and the population of candidate tests consist of 100 individuals.

To compare the quality of the solutions returned by EEA and MOOA, we perform a post-evaluation, in which we test the evolved models on $20,000$ examples that have not been used during evolution. In order to analyze the long-time behavior of the algorithms, for a few settings, we allow the algorithms to query $1,200$ additional labels of input examples (see Table 2). In the remaining settings, the algorithms only query the labeling oracle 100 times. Altogether, we did $8,600$ evolutionary runs as outlined in Table 2.

A single evolutionary run executes two phases in an alternating order: (A) evolving candidate tests and (B) evolving candidate models. Starting the evolutionary run, the population of candidate models is initialized randomly. Each time the set of input examples is augmented with an additional test labeled by the oracle, the population of candidate models is reinitialized with random individuals, preserving the best candidate models. Before each execution of phase (A), the population of candidate tests is (re-)initialized with random individuals. Both phases evolve their populations for only 5 generations in order to avoid too specialized individuals. Preliminary experiments showed that using more generations leads to overfitting to the sparse data. The best models evolved in (B) form a committee that assesses the fitness of tests. The best candidate test evolved in (A) is taken as a query to the labeling oracle, after which the set of input examples is augmented accordingly. In an actual application of our approach, this would correspond to asking the user for feedback on the corresponding sequence diagram.

The post-evaluation, needed solely for evaluation purposes, is computationally costly for long evolutionary runs of, e.g., more than 100 queries. Therefore, we picked six representative settings allowing for $1,200$ queries. Since $1,200$ queries are not feasible in our scenario, these settings are solely for benchmarking purposes. In the remaining $8,000$ evolutionary runs we allowed for 100 queries only. Each combination of maximum number of states and alphabet size was sampled 50 times. The performance of the best individuals of the last generation for each setting was tested for significance using the Mann-Whitney U Test (MWU).

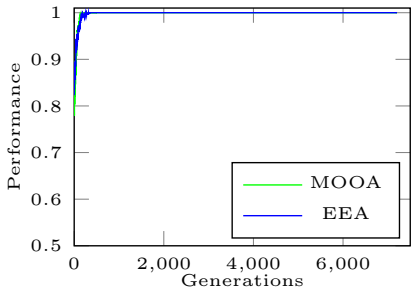
3.2 Results

We start our evaluation with binary alphabets and a maximum number of 4, 8 and 32 states as shown in Figs. 2a, 2b and 2c. The figures show the performance on a test set of the best individual in evolutionary runs allowing for $1,200$ queries.

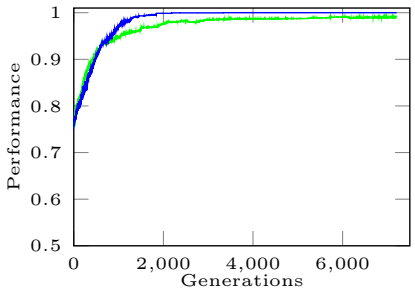
For the relatively simple problem of $|Q| = 4$, $|\Sigma| = 2$ (Fig. 2a) the performance of the two algorithms is nearly the same (MWU p-value$= 1$). For $|Q| = 8$, $|\Sigma| = 2$ (Fig. 2b) there is a non-significant tendency in favor of EEA ($p = 0.23$). However, for more complex problems, for example $|Q| = 32$, $|\Sigma| = 2$ (Fig. 2c), MOOA significantly outperforms EEA ($p < 10^{-15}$). Especially, the oscillations in the performance of EEA indicate that it loses good solutions repeatedly, while MOOA keeps them in the population.

Next, we present results for larger alphabets. In Figs. 2d, 2e and 2f significant improvements of MOOA over EEA can be recognized. In addition, in Fig. 2f, we also show the actual fitness values (dashed lines) that are available to the evolutionary algorithm at runtime based on the provided training data. In the beginning, these fitness values are close to 1, as it is simple to find DFAs that are consistent with the small set of input examples present in the beginning. A high fitness is therefore easily achieved. With increasing number of queries, the set of examples grows, the inference problem gets more difficult, and the fitness approaches the actual test performance over time, as expected. In the setting of Fig. 2d, MOOA converges faster to a test set accuracy of 1 than EEA does. In more difficult settings, MOOA outperforms EEA, as it can be seen in Figs. 2e and 2f. Note that the matrix T has the same size of 64 entries in the Figs. 2c and 2f. While EEA shows approximately the same performance in Fig. 2f and Fig. 2c, MOOA takes advantage of the different structure of the search space and clearly performs better. The oscillations in the performance for this complex setting indicate again that EEA loses good solutions repeatedly. Furthermore, the queries evolved by MOOA seem to have a greater effect on the model performance, making the increase in performance faster with MOOA.

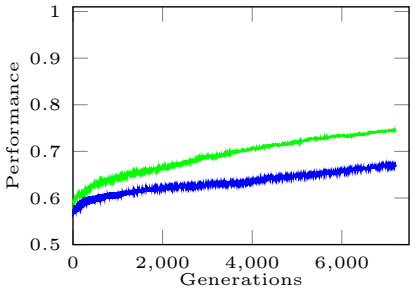
In order to provide a broader overview on the performance differences between MOOA and EEA, Fig. 3 shows a heat map of these differences. The performance difference here denotes that the difference between MOOA's and EEA's best fitness of the last generation is calculated. For a performance difference greater than 0, the heat map shows an advantage of MOOA over EEA on the test set. If, for instance, the performance difference is 0.1, MOOA classified an extra of $2,000$ examples correctly. Due to the computational costs, this map only shows a profile for evolutionary runs for 100 queries. Especially for more difficult problems, this snapshot is taken at an early stage. The red line in Figs. 2d, 2e and 2f indicates the spot where the evolutionary runs for the heat map stopped. The heat map shows the performance difference for the settings from 3 states up to 12 states, with alphabet sizes of 3 up to 10. For problems with either a rather small number of states or a smaller alphabet, EEA and MOOA perform, as already indicated above, competitively. In some settings, there is a tendency in favor of MOOA. As the number of states and the size of the alphabet increases, a significant improvement of MOOA over EEA can be recognized. This significant improvement is indicated by the orange and red colors in the heat map showing a difference in test set accuracy from 0.08 up to 0.12, i.e. MOOA was more accurate on up to 12% of the test set examples.



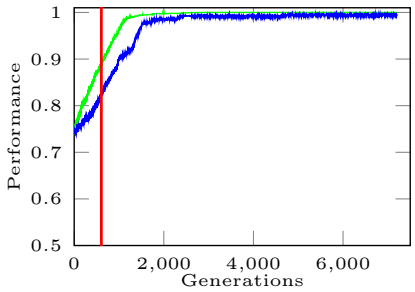
(a) $|Q| = 4$ and $|\Sigma| = 2$ ($p = 1$)



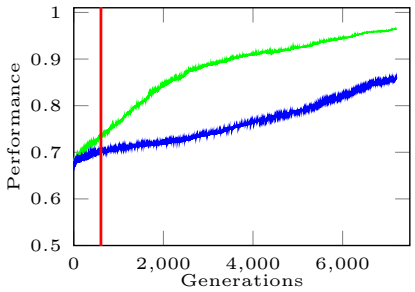
(b) $|Q| = 8$ and $|\Sigma| = 2$ ($p = 0.23$)



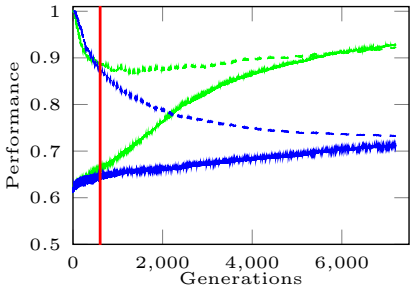
(c) $|Q| = 32$ and $|\Sigma| = 2$ ($p < 10^{-15}$)



(d) $|Q| = 4$ and $|\Sigma| = 7$ ($p = 0.6818$)



(e) $|Q| = 7$ and $|\Sigma| = 7$ ($p < 0.0019$)



(f) $|Q| = 8$ and $|\Sigma| = 8$ ($p < 10^{-22}$)

Figure 2: Evolutionary runs with $1,200$ queries, shown is the averaged test set accuracy (Performance) of best individuals over generations from 50 independent evolutionary runs for different combinations of alphabet size $|\Sigma|$ and number of states $|Q|$.

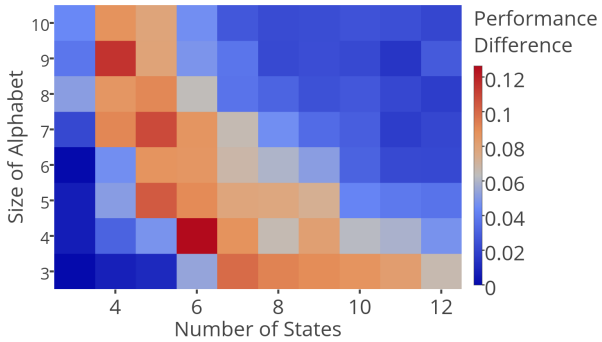


Figure 3: Performance difference between MOOA and EEA, a positive value means MOOA leads (evolutionary runs limited to 100 queries).

For more difficult settings, the difference at this early stage of the evolution decreases, since the amount of training data is not sufficient to yield a bigger difference. However, Figs. 2e and 2f indicate that these fields are expected to become red if more queries are allowed.

4 Discussion

Our approach MOOA is definitely competitive to the state-of-the-art approach EEA. While for small problem sizes ($|Q| \leq 8$, $|\Sigma| = 2$), either no difference in performance is observed or there is a tendency in favor of EEA, for bigger problem sizes, such as $|Q| = 32$, $|\Sigma| = 2$ or $|Q| \geq 4$, $|\Sigma| > 6$, MOOA achieves higher performance in an early stage of the evolutionary run. It is expected that EEA would at least catch up in longer evolutionary runs in the case of bigger problems as well. However, longer evolutionary runs here also mean that there is more interaction with the user (in the form of requests to label words). As is well known from interactive evolution, the number of requests to the user need to be minimized to avoid that the user becomes demotivated. Therefore, the number of queries is to be strictly limited, making the faster performance increase of MOOA an advantage over EEA in our problem domain of search-based software engineering. Also, the better performance for bigger alphabets ($|\Sigma| > 2$) is an advantage in our domain because the alphabet corresponds to the number of involved operations in the requirements specification, which is expected to be much bigger than two (cf. Fig. 1, Table 1). The overview shown in Fig. 3 shows a blue triangle on the top right hand side ($|Q| + |\Sigma| \geq 16$). This can be traced back to the fact that the evolutionary runs for 100 queries were too short to reveal the difference between MOOA and EEA that would be expected for later generations. As seen especially in Fig. 2f, the difference in the test set accuracy increases over time after an initial period where both perform about equally. Increasing the number of queries, we expect more and more of the blue area ($|Q| + |\Sigma| \geq 16$) to become red (i.e., performance lead of MOOA).

The overall experiment setup has a limitation as we are using two different evolutionary algorithms for MOOA and EEA. This was, however, necessary, because there are big conceptual differences between the two approaches. EEA uses only a single objective, but MOOA is a multi-objective approach that re-

quires special treatment regarding the fitness function and selection in the evolutionary algorithm. With NSGA-II, we use a standard tool and allow for simple reproducibility. In the case of EEA, we closely follow the description given by Bongard and Lipson (2005) to ensure a meaningful comparison. Hence, we are comparing not only the qualitatively different approaches of MOOA and EEA to the grammatical inference problem itself but also their different evolutionary algorithms at the same time. This complicates determining the cause of the significant difference between them. However, a trivial cause, such as a mere difference in the mutation rate, can be excluded because both approaches use an almost minimal mutation rate of changing one entry in the transition matrix at a time. From the point of view of our application of requirements specification-by-example, the results are significant and, hence, clearly indicate an advantage of MOOA over EEA.

5 Conclusion

We have shown how requirements elicitation can be automated using coevolution in an active learning approach that proposes additional training data during the evolutionary run. We have reduced the problem to grammatical inference. For relevant input alphabet sizes of three and greater, we significantly outperform the state-of-the-art method (Bongard and Lipson, 2005) by our strictly multi-objective approach in both evolutionary processes of the coevolution. For minimal input alphabet sizes of two, there is a tendency to outperform the state-of-the-art method and the difference gets bigger with increasing automaton size. Distinctive features are the sparse-data and the interactive evolution approach. In contrast to the classical grammatical inference problem, we can request only a few examples from the user. The coevolutionary approach helps to ask the user the right questions during the process and guides the user to provide more relevant examples as well.

In future work, we hope to extend our approach to more complex sequence diagrams in order to provide more functionality to expert users, and to statecharts instead of finite automata. Other options are the inclusion of non-functional requirements on the requested services (e.g., maximal computing time, minimal required reputation of services) and acquiring additional information during the process from other search requests that have been sent to the dynamic software service market as an extension to the active learning approach. In order to make our approach practically applicable, the number of queries required to obtain good solutions needs to be brought down even further. To this end, we will investigate the option of adding more objectives regarding structural aspects (e.g., for recognizing loops, prefixes or suffixes) and study the trade-off between minimizing the interaction and minimizing the overall computing time of the evolutionary run while keeping results of the same quality.

References

D. Angluin. 1987. Learning Regular Sets from Queries and Counterexamples. *Inf. Comput.* 75, 2 (1987), 87–106. DOI:http://dx.doi.org/10.1016/0890-5401(87)90052-6

J. C. Bongard and H. Lipson. 2005. Active Coevolutionary Learning of Deterministic Finite Automata. *Journal of Machine Learning Research* 6 (2005), 1651–1678. http://www.jmlr.org/papers/v6/bongard05a.html

C. de la Higuera. 2005. A bibliographical study of grammatical inference. *Pattern Recognition* 38, 9 (2005), 1332–1348. DOI:http://dx.doi.org/10.1016/j.patcog.2005.01.003

C. de la Higuera. 2010. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, New York, NY, USA.

K. Deb. 1998. An Efficient Constraint Handling Method for Genetic Algorithms. In *Computer Methods in Applied Mechanics and Engineering*. 311–338.

K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evolutionary Computation* 6, 2 (2002), 182–197. DOI:http://dx.doi.org/10.1109/4235.996017

M. Faunes, H. Sahraoui, and M. Boukadoum. 2013. Genetic-Programming Approach to Learn Model Transformation Rules from Examples. In *Theory and Practice of Model Transformations: 6th International Conference, ICMT 2013, Budapest, Hungary, June 18-19, 2013. Proceedings*, K. Duddy and G. Kappel (Eds.). Springer, Berlin, Heidelberg, 17–32. DOI:http://dx.doi.org/10.1007/978-3-642-38883-5_2

E. M. Gold. 1967. Language Identification in the Limit. *Information and Control* 10, 5 (1967), 447–474. DOI:http://dx.doi.org/10.1016/S0019-9958(67)91165-5

J. Gómez. 2006. An Incremental-Evolutionary Approach for Learning Deterministic Finite Automata. In *IEEE International Conference on Evolutionary Computation, CEC 2006, part of WCCI 2006, Vancouver, BC, Canada, 16-21 July 2006*. 362–369. DOI:http://dx.doi.org/10.1109/CEC.2006.1688331

D. Harel, H. Kugler, and A. Pnueli. 2005. Synthesis Revisited: Generating Statechart Models from Scenario-Based Requirements. In *Formal Methods in Software and Systems Modeling*. LNCS, Vol. 3393. Springer Berlin Heidelberg, 309–324. DOI:http://dx.doi.org/10.1007/978-3-540-31847-7_18

M. Harman, S. A. Mansouri, and Y. Zhang. 2012. Search-based software engineering: Trends, techniques and applications. *ACM Comput. Surv.* 45, 1 (2012), 11. DOI:http://dx.doi.org/10.1145/2379776.2379787

G. Kappel, P. Langer, W. Retschitzegger, W. Schwinger, and M. Wimmer. 2012. Model Transformation By-Example: A Survey of the First Wave. In *Conceptual Modelling and Its Theoretical Foundations*, A. Düsterhöft, M. Klettke, and K.-D. Schewe (Eds.). Springer, Berlin, Heidelberg, 197–215. DOI:http://dx.doi.org/10.1007/978-3-642-28279-9_15

M. Kessentini, H. Sahraoui, M. Boukadoum, and O. Ben Omar. 2012. Search-based model transformation by example. *Software & Systems Modeling* 11, 2 (2012), 209–226. DOI:http://dx.doi.org/10.1007/s10270-010-0175-7

T. Kühne, H. Hamann, S. Arifulina, and G. Engels. 2016. Patterns for Constructing Mutation Operators: Limiting the Search Space in a Software Engineering Application. In *Applications of Evolutionary Computation (EvoApplications 2016) (LNCS)*, Vol. 9594. Springer, 278–293.

B. Lambeau, C. Damas, and P. Dupont. 2008. State-Merging DFA Induction Algorithms with Mandatory Merge Constraints. In *Grammatical Inference: Algorithms and Applications, 9th International Colloquium, ICGI 2008, Saint-Malo, France, September 22-24, 2008, Proceedings*. 139–153. DOI: http://dx.doi.org/10.1007/978-3-540-88009-7_11

K. J. Lang, B. A. Pearlmutter, and R. A. Price. 1998. Results of the Abbadingo One DFA Learning Competition and a New Evidence-Driven State Merging Algorithm. In *Grammatical Inference, 4th International Colloquium, ICGI-98, Ames, Iowa, USA, July 12-14, 1998, Proceedings*. 1–12. DOI:http://dx.doi.org/10.1007/BFb0054059

S. M. Lucas and T. J. Reynolds. 2003. Learning DFA: evolution versus evidence driven state merging. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2003, 8 - 12 December 2003, Canberra, Australia*. 351–358. DOI:http://dx.doi.org/10.1109/CEC.2003.1299597

S. M. Lucas and T. J. Reynolds. 2005. Learning Deterministic Finite Automata with a Smart State Labeling Evolutionary Algorithm. *IEEE Trans. Pattern Anal. Mach. Intell.* 27, 7 (2005), 1063–1074. DOI:http://dx.doi.org/10.1109/TPAMI.2005.143

D. Le Ly and H. Lipson. 2014. Optimal Experiment Design for Coevolutionary Active Learning. *IEEE Trans. Evolutionary Computation* 18, 3 (2014), 394–404. DOI:http://dx.doi.org/10.1109/TEVC.2013.2281529

E. Mäkinen and T. Systä. 2000. An Interactive Approach for Synthesizing UML Statechart Diagrams from Sequence Diagrams. In *Proceedings of OOPSLA 2000 Workshop: Scenario based round-trip engineering*. 7–12.

L. van Rooijen and H. Hamann. 2016. Requirements Specification-by-Example Using a Multi-Objective Evolutionary Algorithm. In *Third Int. Workshop on Artificial Intelligence for Requirements Engineering (AIRE'16), Beijing, China*. http://dx.doi.org/10.1109/REW.2016.015

B. Settles. 2012. *Active Learning*. Morgan & Claypool Publishers. DOI:http://dx.doi.org/10.2200/S00429ED1V01Y201207AIM018

F. Tsarev and K. Egorov. 2011. Finite state machine induction using genetic algorithm based on testing and model checking. In *13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Companion Material Proceedings, Dublin, Ireland, July 12-16, 2011*. 759–762. DOI:http://dx.doi.org/10.1145/2001858.2002085