Focusing Learning-based Testing away from Known Weaknesses

Christian Fleischer and Jörg Denzinger

Department of Computer Science, University of Calgary, Canada
{crfleisc,denzinge}@ucalgary.ca

Abstract. We present an extension to learning-based testing of systems for adversary-induced weaknesses that addresses the problem of repeated generation of known weaknesses. Our approach adds to the normally used fitness measure a component that computes the similarity of a test to known tests that revealed a weakness and uses this similarity to penalize new tests. We instantiated this idea to the testing of ad-hoc wireless networks using the IACL approach, more precisely to applications in precision agriculture, and our experiments show that our modification results in finding substantially different tests from the test(s) that we want to avoid.

1 Introduction

In the last decade, search-based software engineering has shown substantial success in supporting various kinds of testing of systems (see, for example, [1, 2]). But in this time we also have seen an increased need to test distributed applications, especially for potential abuses by adversarial users with all kinds of agendas. While there are also approaches for supporting these kinds of testing problems using evolutionary learning to identify possible weaknesses (see [3–5]), a problem limiting the use of these testing approaches is that repeated runs of the testing systems often do not result in identifying many possible weaknesses without fixing previously found weaknesses first. Naturally, due to the use of random factors in the search, not every run of such a testing system will have the same result, but depending on how strongly the used fitness function identifies a particular weakness of the tested system, many runs can identify the same weakness over and over again. And, after a risk analysis, a found weakness might be deemed as unnecessary (or too expensive) to fix and then naturally the testing system needs to be made aware that this particular test is not considered a weakness anymore.

In this paper, we present an approach that solves this problem of learning-based testing. The basic idea is to enhance the used fitness function to allow specifying previously found weaknesses of the tested system and to penalize individuals created by the evolutionary search based on their similarity to these specified weaknesses (in the form of individuals from previous runs). Due to this intended use of the similarity it concentrates on the difference in the effects an

attack has on the tested system, considering both the components of the tested system and its environment. We instantiated this general idea for the IACL approach to test ad-hoc wireless networks that was presented in [5, 6].

Our evaluation of the improved testing system showed that our method finds rather different solutions (with respect to the behavior of the tested system) when given a single source solution to avoid. Iterating the process, i.e. adding the newly created best solution to the set of solutions to avoid, also rather consistently creates solutions that differ from all solutions that should be avoided.

The paper is organized as follows: After this introduction, in Section 2 we present the basic IACL approach. Following that, in Section 3, we present our modification of IACL that allows it to focus away from already known weaknesses. In Section 4, we describe the instantiation of IACL and our modification to the application area of precision agriculture. Section 5 presents the experimental evaluation using the instantiation. In Section 6, we relate our work to the known literature and Section 7 concludes with some remarks on future work.

2 The IACL approach

Incremental Adaptive Corrective Learning (IACL, see [5]) is a search-based testing method for ad-hoc wireless networks. In the following, we will present how we model a particular test as an individual, then briefly describe the evolutionary learning method that zooms in more and more on tests fulfilling a given test goal and then describe how a given test is evaluated, including how we correct an individual in order to not obviously violate network protocol requirements.

2.1 General set-up

An ad-hoc wireless network consists of a collection of mobile wireless nodes (also called agents) within a geographic area. They exchange messages following several network protocols that specify requirements for these messages. Usually these requirements include obligations which result in additional messages being spread through the network.

A particular test of an application of such a wireless network requires defining a scenario, which describes the geographical area of the network and behaviors for each of the nodes. These behaviors can be subdivided into its movements during a given period of time and its communications during that period of time. More formally, a scenario $S = (geo, t_{max}, vel_{max}, M, C)$ consists of the geographical area geo, the length t_{max} of the test run, the maximal velocity vel_{max} of any node, the set M of protocols used and the set C of so-called customer nodes that are acting within the network.

A particular test adds to S a set A of attack agents (nodes) that are also acting within the given scenario. Nodes in both A and S perform actions out of two sets Mov and $Comm$ and their behavior is characterized by an action sequence for each of them. The set Mov of movement actions consists of actions either of the form $spos$ or $v = (t, head, vel)$ where $spos$ is a starting position for

the node. v is a movement change action, where t is a real number, $0 \leq t \leq t_{max}$, *head* is a natural number, $0 \leq head \leq 359$, giving a direction heading and *vel* is a real number, $0 \leq vel \leq vel_{max}$, giving a velocity, altogether indicating a change in direction and velocity at a given point in time. The set $Comm$ has elements of 5 different forms: $c \in Comm = s|p|in|dr|mo$, where $s = (t, dest, data)$ is a data initiation action, indicating that the node should initiate the procedure necessary to send *data* to *dest* at time index t. $p = (t, target)$ is a so-called protocol action, which is the result of obligations a node has due to the actions of other nodes. *target* is the protocol-induced message sent at t. s- and p-actions are sufficient to describe the communications of customer nodes.

The other types of actions are used by attack agents to compromise the network. $in = (t, target)$ represents an *insertion action* where the agent should insert the packet specified in *target* into the network at time t. *Drop actions*, $d = (t, template)$, indicate to an agent that it should ignore a protocol-induced obligation of the form *template*, which should be sent at t. Finally, *modification actions*, $mo = (t, t_{actual}, template, target)$, indicate to an agent that a protocol-induced packet, *template*, normally sent at t should be modified by replacing it with *target* and sending it at t_{actual}.

In our testing system we are searching for a particular test for a given scenario, which means that we are looking for action sequences for the attack agents. We will use AS^i_{cust} to refer to the action sequence of customer node i and AS^j_{att} for the sequence of attack agent j. In general, for a node j an action sequence has the form $a^j_1, \ldots, a^j_n$.

2.2 The IACL main loop

As usual for an evolutionary algorithm, the main loop of IACL creates generations of individuals that each represents a test, more precisely action sequences for the attack agents in A. In order to deal with the very large set of possible individuals, IACL starts with a small number of attack agents (two) and increases the number over time (hence the incremental in the name). Whenever a new attack agent is added, the search first concentrates on the new agent to adapt its behavior to the behaviors of the already existing agents.

More precisely, IACL first creates n_{pop} tests with two attack agents and action sequences consisting just of a starting position for these agents. After evaluating these tests (which usually results in them being altered, see next subsection) using a given fitness function that reflects the test goal(s), we keep the n_{surv} best tests and replace all other tests with new ones created by applying various genetic operators (for more information on the operators, see [7]). This process of creating new generations is repeated ng_{GL_1} times but may be stopped early, if for n_{Stall} iterations the fitness of the best test does not improve.

After this first general learning phase, the method loops through the following routine (we describe the i-th loop of n_{max}). First, an attack agent is added, resulting in adding an action sequence for this agent to each test that consists just of a start position. In an adaptive learning phase AL_i we create new sets of tests like described above ng_{AL_i} times but apply genetic operators only to the

action sequence of the new attack agent (with the same condition for stalling as above). Then we have a general learning phase GL_i for ng_{GL_i} generations (with the same stalling condition) in which the genetic operators now target the action sequences of all attack agents.

2.3 Evaluating (and correcting) an individual

In general, an individual (i.e. a test) is evaluated by running it, together with the given scenario, in a simulation and collecting information metrics that are used by the fitness function(s). But, due to the various requirements of the used network protocols, a newly created individual usually is easily identifiable as not conforming with the requirements by the customer nodes, which naturally is not what an adversary in control of attack agents would do. Therefore, in order to test for realistic attacks, we correct the individuals while doing the simulation. The performed corrections aim at fixing only unintended problems of an individual, while preserving the intentional violations of the requirements that were introduced in ancestors of an individual (or in the creation of the individual itself) and that are not easily detectable by non-attack nodes in the network.

With IACL, at the start of the simulation, each agent/node is placed at the starting position *spos* of its action sequence. Using the time indexes of each action in each sequence we move the simulation of the test represented by the individual forward. If we take the attack agent j and assume that $a_1^j, \ldots, a_{i-1}^j$ are the actions in its action sequence that have already been performed and further assume that the time index for action a_i^j has arrived, then there are the following three major cases (with some sub-cases) to consider:

Case1: no protocol-induced obligation (due to receiving a message or another stimulus) needs to be fulfilled:
If $a_i^j \in Mov$, then j simply performs a_i^j and AS_{att}^j remains unchanged. If $a_i^j \in Comm$ we have the following sub-cases, depending on the form of a_i^j.

- form s: agent j starts the execution of the action according to the protocols in place and AS_{att}^j remains unchanged
- form in: j transmits the packet indicated in in and AS_{att}^j remains unchanged
- form mo or dr: AS_{att}^j needs to be corrected, since there is no packet to modify or drop. a_i^j is therefore removed from AS_{att}^j
- form p: a_i^j is an artifact from a previous test idea which is no longer required. a_i^j is removed from AS_{att}^j, node j moves ahead to a_{i+1}^j and the simulation continues.

Case 2: there is a protocol-induced obligation (requiring to send a packet):
As before, if $a_i^j \in Mov$, then j performs a_i^j, but j then also performs the protocol-induced obligation. And this protocol-induced obligation is inserted into AS_{att}^j as a_{i+1}^j as a p action. j moves ahead to a_{i+2}^j in AS_{att}^j and the simulation continues. If $a_i^j \in Comm$ then we do the following, depending on the form of a_i^j:

- form s or in: node j performs a_i^j and then the protocol-induced action is inserted into AS_{att}^j as a_{i+1}^j. j moves ahead to a_{i+2}^j (if it exists) and the test continues.
- form mo or dr: the agent checks to see if the packet listed in the *template* area of a_i^j matches the protocol-induced action that the agent is required to take. Matching involves the agent checking to see if the packets in a_i^j's template area and the protocol-induced action are of the same type, and that their fields contain the same information. If the packets match, then the protocol-induced obligation is not performed, and instead a_i^j is executed (this is an intentional break of the obligation). If the packets do not match, then there has been a change in the network's behaviour (compared to a previous test) that caused a_i^j to be invalid. a_i^j is replaced in AS_{att}^j with a p action corresponding to the actual protocol-induced obligation, the agent then fulfills this obligation and the simulation continues.
- form p: If the network-induced obligation matches (see above) the target area of a_i^j, then the agent fulfills the obligation and the test continues. If it does not match, then a_i^j is replaced with a new p action reflecting the actual protocol-induced obligation required of the agent, the agent fulfills the obligation and the simulation continues.

Case 3: the test has reached a point where the node j has a new network-induced obligation to fulfill but it has no element a_i^j at the current time index:
In this case, the agent examines its AS_{att}^j in order to find a_{ins}^j and a_{ins+1}^j, where the time index of a_{ins}^j is earlier than the current time index and a_{ins+1}^j's time index is greater than the current time index. A new p action corresponding to the network-induced obligation is created and inserted into AS_{att}^j between a_{ins}^j and a_{ins+1}^j and the simulation continues.

While the above is the behavior of an attack agent the procedure is the same for customer nodes, except that customer nodes naturally do not use in, mo or dr actions.

2.4 Other IACL components

The two remaining components of IACL not explained so far are the set of genetic operators and the fitness function(s). Due to the complexity of network protocols, a large number of genetic operators are needed, together with a strategy on how to decide what operators should be applied when. Each field in a packet is the target of at least one mutation operator, but for quite a number of fields there are more such operators (using knowledge about the role of the field in the protocol). Other mutations insert actions into action sequences or change communication actions into drop actions. It is also possible to fragment messages (via such a mutation operator). Additionally, there are crossover operators that apply to different levels of a test, switching parts of sequences between different agents within the same test, between agents in different tests, or switching whole attack sequences between tests. A detailed description of the various operators and how they are controlled is given in [7].

Naturally, the chosen fitness function is very important for finding tests that fulfill a test goal, i.e. tests that find a particular kind of weakness of the tested system. While usually there is an obvious metric that measures the fulfillment of a particular goal by a test (resp. its simulation), as [6] (and for a totally different kind of system [4]) showed, additional metrics are needed to guide a learning-based testing system toward tests that fulfill the given goal. To deal with this need for combining metrics and the fact that different test goals require different metric combinations [5, 6] used fitness functions that essentially computed the weighted Euclidean distance between the position of an individual in the n-dimensional space created by n metrics and a so-called goal point in the same space which represents the "optimal" combination of metric values for the test goal. While there are some rather general metrics, many of the metrics are application- and some test goal-dependent.

3 Modifying IACL to avoid known solutions

Since evolutionary algorithms at their core are optimization methods, it is not exactly surprising that performing several runs of such an algorithm for a given problem instance does not always result in producing different solutions. In fact, some proponents of other optimization methods consider the fact that two runs are not always producing the same result a weakness of evolutionary methods. But from the perspective of security testing, getting the same solutions in several runs naturally represents a weakness because resources are essentially "wasted". And the longer a run takes to produce solutions, the more serious the problem becomes.

While tricks like varying parameters between runs might reduce this problem, they offer no guarantee that runs might not still produce the same solutions, even if there are other solutions of interest. Our solution approach to this problem is to modify the used fitness function with a component that penalizes solutions based on their similarity to previously found solutions.

More precisely, given a basic fitness function fit_{base} and a set $Sol = \{sol_1, ..., sol_k\}$ of known solutions, we evaluate an individual ind using the function fit_{no-sim} defined as follows:

$$fit_{no-sim}(ind, Sol) = fit_{base}(ind) + sim_{Set}(ind, Sol) * fit_{base}(ind) * w_{sim} \quad (1)$$

(this assumes that fit_{base} is intended to be minimized.) w_{sim} is a parameter that determines the influence that similarity to known solutions has on the evaluation of an individual. sim_{Set} should combine the similarities sim of ind to each of the elements of Sol. Obviously, there are several possible ways to achieve this. We used the following definition:

$$sim_{Set}(ind, Sol) = 1/k * \sum_{i=1}^{k} sim(ind, sol_i) \quad (2)$$

While the previous definitions were not IACL specific, defining sim naturally has to reflect that IACL has the attack agents test a scenario happening in an

environment and a system consisting of customer agents. Therefore, like fit_{base}, sim combines several measures from two groups, namely measures comparing events related to the environment and events that happen to customer agents. Assuming that ind is already corrected (as described in Section 2.3), the second group of events can be evaluated by just looking at actions/messages recorded for them, while the first group requires analyzing the simulations of both individuals.

Both groups of events potentially contain several different types of events. Two individuals can be compared based on the number of occurrences of the event, as well as the differences in times that those events occur between the two individuals. In fact, we combine these two ideas for most of our measures that we combine to create sim.

More precisely, a dual event similarity ev_{sim} is defined as

$$ev_{sim}(ind_1, ind_2) = \sum_{ag \in Ag} occ_{ev}(ind_1, ind_2, ag) * time_{ev}(ind_1, ind_2, ag) * w_{ev}(ag) \tag{3}$$

where Ag is either A or C (although we use in our instantiation only agents from C), occ_{ev} is the difference of the number of occurrences of the event in the action sequence for agent ag in the two individuals. $time_{ev}$ is the minimal sum of time differences between assigned events between the two action sequences for ag of all possible assignments of events of the type in ind_1 to events in ind_2 such that sequence is preserved (and the assignment's length is the smaller of the numbers of occurrences of the event type in the two action sequences for ag) divided by t_{max} and subtracted from 1. Note that by making w_{ev} potentially different for different agents we can focus on particular agents by providing them with a higher weight.

A small conceptual example for the computation of ev_{sim} is the following. Let us assume that ind_1 has created the action sequences $((a_{11}, a_{12}, b_{13}), (c_{21}, d_{22}))$ for two customer nodes and ind_2 has created $((a'_{11}, b'_{12}), (c'_{21}, d'_{22}))$ (for the same nodes) with a_{11}, a_{12}, a'_{11} being of the same event type and a_{11} happening at time 2, a_{12} at time 4 and a'_{11} happening at time 5 (and $t_{max} = 10$). If our event is occurrences of actions of type a, then for the second agent occ_{ev} is 0 and consequently $time_{ev}$ is also 0. For the first agent we have two possible assignments of a-events between the two individuals, namely a'_{11} to a_{11} and a'_{11} to a_{12}. Since the time difference for the first assignment is the sum consisting of 3 and for the second assignment is 1, the second assignment determines $time_{ev}$ as $1 - 1 \div 10$, which together with the occurrence difference of 1 produces as ev_{sim}-value 0.9 (assuming that w_{ev} for the first agent is also 1).

We combine a set of event similarities (which can be dual as defined above, but also computed differently, see the next section) $ev_{sim,1}$,...,$ev_{sim,l}$ by summing their evaluations up:

$$sim(ind_1, ind_2) = \sum_{j=1}^{l} ev_{sim,j}(ind_1, ind_2) \tag{4}$$

Note that due to the various weight parameters, the maximal similarity, i.e. between identical solutions, is usually not equal to 1 (which obviously is not a requirement for our application).

4 Instantiation to precision agriculture

For our evaluation of the proposed modification of the fitness computation in IACL from Section 3 we have chosen the same application as in [5], namely precision agriculture. The idea behind this application area is to use wireless sensor networks monitoring growing conditions in a field together with actuators that can influence these growing conditions (like watering an area of the field) and that are also parts of the wireless network. By so automating the care for crops the aim is to reduce the amount of manual labour of farmhands and the overall operating costs. Since all nodes require batteries for operation, one test goal determines if it is possible to deplete these batteries much faster than normal, so that the intended gains are negated. Naturally, another test goal is to make sure that the right actuators act correctly in all conditions.

The simulation uses as M the Internet Protocol (IP, [8]), the User Datagram Protocol (UDP, [9]), Ad-hoc On-demand Distance Vector Algorithm (AODV, [10]) for message routing and an improvised agriculture network application layer protocol (IAP) as defined in [5]. This means that *template* and *target* from Section 2.1 have the form $(IP, UDP, Payload)$, where IP and UDP represent all the fields that these protocols require and $Payload$ contains as possible packets all the packets from AODV and the *watering trigger* and *watering request* packets from IAP. We will refer to the packets from IAP as DATA packets (or messages or actions). Among the packets from AODV are the route request (RREQ), route reply (RREP) and route error packets (RERR), which are used to find routes between two agents (RREQ to start the search, RREP to report the results of the search) and to indicate that a route is not valid anymore (RERR). In order to use these protocols, an agent needs to keep a so-called routing table that aims to represent the knowledge of how to reach a particular agent. An invalid routing table usually results in a lot of messages that try to make it valid again.

For fit_{base} we use, as in [5], as metrics the power consumption of the whole network determined by counting the number of transmissions made, the distribution of this power consumption over the course of the simulation, the number of nodes that have exhausted their power supply before the end of the simulation, the number of sensor nodes that have moisture levels below the acceptable threshold at the end of the simulation and how long each sensor node had a moisture level below the lowest level measured in the network with no attack agents present.

For constructing sim_{Set}, we used the following dual event similarities. The event similarities in the group of events happening in the environment are measuring the events that the soil moisture dropped below a given measure ($sim_{ev,moi}$, we measure moisture, as in fit_{base}, where the sensor nodes are located, so that this could also be considered as a measure in the second group

but conceptually we could do this measurement everywhere in the simulated environment). Events about the customer nodes are a customer node sending a RREQ message ($sim_{ev,rreq}$), a RREP message ($sim_{ev,rrep}$), a RERR message ($sim_{ev,rerr}$) or a DATA message ($sim_{ev,sdata}$) and the events that a customer node received a DATA message ($sim_{ev,rdata}$)[1].

Additionally, we used $sim_{ev,en}$ which is the difference in the total amount of energy used throughout the two simulations (which is obviously in the group of events related to the customer nodes) weighted by a $w_{ev,en}$ and $sim_{ev,t-out}$ which is the time difference between the two simulations when a customer node runs out of power (summed-up over all customer nodes, with the end of the simulation being the used time if a node did not run out of energy, weighted by $w_{ev,t-out}$).

Similar to some of the measures in the dual event similarities, we have an additional event similarity $sim_{ev,maxseq}$ that is applied to the customer nodes. For a node, we abstract and filter the action sequences in both individuals to only containing the message types RREQ, RREP, RERR and DATA. We then determine the length of the largest subsequence of those types that both sequences have in common. $sim_{ev,maxseq}$ is this number weighted by a $w_{ev,maxseq}$ for this node.

Finally, event similarity measure $sim_{ev,alltime}$ also looks at the nodes and the times of events (actions) for a node. It sums up the difference of the times the first action is taken, the difference of the times the second action is taken and so on until we run out of actions in one of the action sequences. And the sum for a node ag is weighted by a $w_{ev,alltime}(ag)$.

Note that looking only at the message/packet/action types from the AODV and IAP protocols is justified for this application, since these types reflect the fact that we have wireless networks (with moving agents) and the goals of the application. The application is also highlighted by not just looking at events sending messages but also events representing the reception of a message. While $sim_{ev,en}$ and $sim_{ev,t-out}$ obviously represent looking at the test goals for this application, $sim_{ev,maxseq}$ and $sim_{ev,alltime}$ are rather strong generalizations and therefore represent also a strong push away from already known solutions (provided that their associated weights are not neutralizing them). As our experiments in the next section show, these event similarities are useful.

5 Experimental evaluation

To evaluate our modification of the fitness computation to focus on avoiding known solutions (i.e. weaknesses) while naturally still trying to find weaknesses, we have chosen a test scenario S that offers a lot of possibilities for an adversary to exploit. We have already provided M. geo is 10000 by 10000. t_{max} for the

[1] These kinds of events could also be applied to attack agents which would create a third group of events. As our experiments show, concentrating on environment and customer agents is enough for our application, but other applications and other testing methods might require this third group.

Table 1. Average results for focused runs with single source

Source		Focused runs			
Sol. nr.	fit_{base}	av. fit_{base}	av. sim_{Set}	av. nr. of gen.	av. run time (in min.)
1	16660.53	16875.55	0.34	90.2	96.9
2	16690.00	16515.56	0.30	119.4	87.4
3	16310.53	16355.63	0.32	84.6	125.3
4	17615.92	17227.34	0.28	83.4	55.3
5	16507.93	17121.76	0.32	25.1	49.0

Table 2. Variance of fit_{base} and sim_{Set} for focused runs with single source

Source		Focused runs			
Sol. nr.	fit_{base}	min fit_{base}	max fit_{base}	sim_{Set} for min	sim_{Set} for max
1	16660.53	16439.35	17295.63	0.32	0.28
2	16690.00	16136.41	17238.95	0.25	0.29
3	16310.53	16079.28	16947.55	0.30	0.34
4	17615.92	16905.89	17306.03	0.35	0.26
5	16507.93	16695.84	17306.60	0.35	0.27

scenario is 1000 time units and vel_{max} (for the attack agents) is also 1000 (which is the distance travelled in 10 time units). The customer nodes are a master node placed at (4500,4000), an actuator node that waters the field at (5000,4500) and 3 sensor nodes placed at (5000,5300), (5000,6100), and (5000,6900). All these nodes do not move. The communication range is 850. Each node has a battery capacity of 150 and without attack agents the nodes will use between 50 and 75 energy units (to deal with the requests for watering due to having the field dry out over time; we did not have rain in this scenario). Since we wanted to test the ability of our approach to avoid a particular solution we needed to be able to create several source solutions for the scenario (without too many runs of the IACL system without the modification) and therefore we allowed 7 attack agents (which provides enough opportunities for finding tests that reveal weaknesses).

For the various parameters of IACL we used the following values: $n_{pop} = 30$, $n_{surv} = 9$, $n_{Stall} = 20$, $ng_{AL_i} = 75$ and $ng_{GL_i} = 75$. The weights in our modification of the fitness computation were set as $w_{sim} = 0.1$, for all agents ag we used $w_{ev,moi}(ag) = 2/1800$, $w_{ev,rreq}(ag) = 11/1800$, $w_{ev,rrep}(ag) = 3.6/1800$, $w_{ev,rerr}(ag) = 3.6/1800$, $w_{ev,sdata}(ag) = 3.6/1800$, $w_{ev,rdata}(ag) = 3.6/1800$, $w_{ev,maxseq}(ag) = 3/180$ and $w_{ev,alltime}(ag) = 2/180$, and finally we used $w_{ev,en} = 11/1800$ and $w_{ev,t-out} = 11/1800$.

To get some initial solutions to avoid and to establish a baseline, we performed around 50 runs of the original version of our IACL instantiation. The biggest similarity sim_{Set} for two of those solutions was 0.55, the minimal similarity was 0.34 and the average similarity between two of these solutions was 0.43. Of all these solutions, we selected the two with the least similarity and then additionally three that had the least similarity to all already selected ones (named 1 to 5) for our first experimental series. For each of these solutions we ran the modified version 10 times. The results of these experiments are reported

in Tables 1 and 2. As Table 1 shows, the average base fitness of the focused runs by our modified system is not much worse than the base fitness of the source solution to be avoided. In fact, already the average base fitness of the 10 focused runs to source solutions 2 and 4 is better than the base fitness of the source solutions. Consequently, the best of the 10 focused runs for these source solutions have solutions that are substantially better than the source solution. Even more, the best runs for source solutions 1 and 3 are also better (see Table 2). Looking at the worst runs, they are worse than the source solutions they avoided (except for source solution 4), with a maximal base fitness difference of 800. Given that the base fitness values are over 16000 and that by avoiding solutions worse solutions are to be expected, this result is definitely not bad.

Looking more closely at the values of our similarity measure, we see in Table 1 that the average sim_{Set}-values are between 0.28 and 0.34 indicating that we are better, i.e. less similar, that what our baseline achieved. Table 1 shows that the similarity of the best solutions in the 10 runs for each source solution is not always lower than for the worst solution. While this is not unexpected (given that solutions found while avoiding a source solution can be better with regards to fit_{base}), a question that these tables cannot answer is whether our proposed similarity measure really is allowing us to find different weaknesses than those revealed by the respective source solutions. To look into this, we observed the behaviors of all agents in the simulations of the source solutions and the best solutions in each of the focused runs and found that these behaviors were rather different (although between focused runs we naturally saw some rather similar ones).

More precisely, in source solution 1, the attack agents send false route requests from customer nodes for other attack agents and one attacker also sends irrelevant information to the master node. And another attacker sends data to a fictitious attack agent. This results in essentially bombarding all customer nodes with messages that they have to route, which resulted in very quick battery depletion for all but the master node and serious network congestion in general. The best focused run for this source solution has the attackers moving a lot through the field, with one attacker dropping many requests it receives while sending requests as if it is the master node and then sending requests for the master node (totally confusing the routing tables of all agents getting these requests). This attacker also does a lot of modifying of messages it receives. Overall, this solution focuses on disrupting the routing tables of the customer nodes, which leads to preventing the sensor nodes from informing the master and actuator node about the need to water the field which then causes the field to go without water for most of the simulation. This is very different from the source solution.

Source solution 2 has the attackers hitting all customer nodes with a sudden burst of nearly constant RREQ messages for around 10 time units, which results in depleting their batteries. The best solution of a focused run makes extensive use of the possibility to fragment messages (on the level of the UDP protocol) and the attack agents especially target the first sensor node (which is the connection between sensor nodes and the other nodes in this scenario) resulting in depleting

this node's battery even quicker than the batteries of the nodes in the source solution. Again, this is very different from the source attack.

Source solution 3 shows a more drawn out version of the behavior seen in source solution 2, having attack agents producing messages that the customer nodes will in the end reject but that require the use of energy. The attack agents also impersonate the master node and immediately after move around which all results in a very congested network. As a result, we see initially too infrequent watering (while the batteries are still working) and no watering starting when the batteries are depleted. As in source solution 1, the master node's battery maintains power the longest. The best solution of the focused runs is very similar to the best solution of the focused runs for source solution 2, attacking the first sensor node and due to that, interrupting rather quickly the legitimate communication within the network (and leading to watering issues relatively early). Again, both attacks are highlighting different weaknesses of the scenario.

In source solution 4, different attack agents use different strategies, with one agent introducing many watering requests and impersonating the master node, while the other agents move around a lot and drop messages while sending RREQ messages. This results in a little more build-up until the batteries of the actuator node and the sensor nodes are depleted and also the moisture levels are bad later than for the other source solutions (which results in this solution having the lowest fit_{base}-value of all source solutions and allowing for all focused runs to it to have better best solutions). The best solution from the focused runs has the attack agents not attack the battery power of the actuator node at all and only the battery of the first sensor node is depleted at all in the simulation. The key problem produced by this solution is insufficient moisture levels and this is achieved by only two attack agents that compromise the routing tables of the sensor nodes. One of the agents is essentially zipping around the sensor nodes (at full speed) sending a lot of message requests with the other agent making sure that all of these requests reach at least one customer node.

Finally, in source solution 5, one attack agent sends a lot of watering requests directly to the master node which results in depleting the battery of the actuator node rather quickly which then leads to insufficient moisture in the field. The other attackers work on depleting the batteries on 2 sensor nodes via having them relay messages between themselves. In contrast to that, the best solution from the focused runs does not target the actuator battery at all. Like the best solution of a focused run for source solution 2, the first sensor node is the main target for battery depletion, but this happens later in this simulation. Soil moisture is problematic earlier due to attack agents invalidating routing tables and creating network congestion.

As these high-level descriptions show, when avoiding a given source solution our method creates very different attacks, showing that our similarity measure really achieves what we want it to do.

We also performed experiments to see the performance of iteratively applying our modification, i.e. we choose a source solution, run our modified version of IACL, add the best solution from this run to the set Sol, run our modified version

Table 3. Experiments with iterated focused runs

Example1			Example2		
Iteration	fit_{base}	sim_{Set}	Iteration	fit_{base}	sim_{Set}
Initial source	16310.53	—	Initial Source	17283.23	—
After 1st it.	16601.95	0.39	After 1st it.	16640.69	0.39
After 2nd it.	16717.64	0.30	After 2nd it.	15997.61	0.29
After 3rd it.	16926.88	0.34	After 3rd it.	16682.59	0.41
After 4th it.	16849.59	0.37	After 4th it.	16500.23	0.35
After 5th it.	16553.99	0.34	After 5th it.	16082.90	0.29

again and so on for 5 iterations of the modified system. Table 3 reports on two such experiments with the best source solution from the previous experimental series (3 from the previous tables, here Example1) and a not so good source not used so far (Example2). As the table shows, we are still getting solutions with rather good fit_{base}-values. In fact, the second iteration for Example2 has the best fit_{base}-value we observed in all of our experiments. Just looking at the sequence of fit_{base} or the sequence of sim_{Set} does not reveal any general trends except that we get good solutions and that the average similarity between solutions is less than what we saw in the baseline experiments, which is exactly what we wanted to achieve.

But, as with the previous experimental series, a look at the behaviors created in the simulations is of more interest. We have already described the weakness produced by the source for Example1. In the first iteration, the best found solution has 4 attack agents contribute in depleting first the actuator nodes battery and later the sensor nodes' batteries. The best solution of the second round also has mostly 4 attack agents being responsible for depleting the battery of one of the sensor nodes halfway through the simulation but the main effect of the attack agents' behavior is to avoid the field getting watered by intercepting watering requests and moving around a lot. The solution found in the third iteration has 6 attack agents contributing to first depleting the first sensor node and then the actuator node by targeting them with a lot of network traffic. This includes one agent who heavily fragments the messages from other agents. In the fourth iteration, 6 attack agents contribute, again, by attacking the batteries of the first and second sensor node using modifications of the messages of the master node (resulting in checksum errors and the requests not being accepted). Finally, in the last iteration, all attackers contribute to achieving the attack. In contrast to the other solutions, this solution does not attack the batteries but blocks any watering from happening relatively early in the simulation by modifying all messages from the master node to the actuator node and making them invalid. We see also a lot of fragmenting of messages, although this does not result in the depletion of any of the batteries.

In Example2, the initial source solution has 5 attack agents contributing by depleting the actuator node's battery around halfway through the simulation. The methods used are impersonating other agents, moving around to invalidate the routing tables and sending watering requests. In the first iteration, the attack

also has 5 attackers contributing in depleting the batteries of the first and second sensor node early in the simulation. The used methods are moving around while sending messages and impersonating customer agents. In the second iteration, 6 attack agents contribute to the achieved effect, which is to block all watering requests the master sends to the actuator node. In fact, not a single watering by the actuator occurs in the whole simulation which explains the good fit_{base}-value. The third iteration's solution only needs 5 attack agents, again. It depletes the batteries of the first and second sensor node which makes the environmental effects rather similar to the solution from the first iteration, but the used methods to achieve this are rather different, namely faking data fragmentation (i.e. sending a packet that claims that it has more fragments coming, which never happens). This creates difference, but not as much as for many of the other iterations, which is reflected by the rather high sim_{Set}-value (compared to other iterations). The fourth iteration has all 7 attackers contributing to the effects of the attack. The battery of the actuator node is the target and it is depleted early in the simulation. The behaviors of the attackers include all kinds of bad behavior without any clear pattern except for the result for the actuator. The final iteration manages to block all watering requests, but in contrast to the second iteration, this is achieved by depleting the battery of the first sensor just before it would send a watering request and naturally that means also that the requests from the other sensor nodes are not relayed. In the simulation for this solution, the master and actuator node end up with higher battery charges than they have in the simulation without the attackers present.

These two examples show, again, that our similarity measure is achieving its aim, namely creating different attacks to a scenario when used to produce a penalty for being too similar to known solutions. Attacks that have similar elements are producing higher similarity values as intended (and are mostly avoided). It also seems that in order to have a low similarity to the already known attacks, involving more attack agents to create more difficult attacks is a trend.

6 Related work

As mentioned in Section 3, focusing evolutionary algorithms away from a particular solution has not exactly been a focus of research. In fact, any interest in similarity of solutions is usually associated with additional interests, like improving migration in distributed GAs (see [11]) or keeping a higher diversity of solutions during the search (see [12]). In [11], similarity, respectively distance of solutions is used to improve the selection of solutions from a particular generation for sending them to other populations by other search agents. In [12] similarity, resp. again distance from each other, is added as an additional search objective. Both papers differ from our approach in that they are not using distance to given known solutions, but apply the distance/similarity to individuals created during the search. Also, they focus on similarity on the level of the struc-

ture of the individual (which for us would be the action sequences of the attack agents) and not on its evaluation.

If we broaden our look to approaches that try to focus evolutionary search outside of the standard focus on optimizing the given fitness measure, then there are also not many works. An exception is [13]. In contrast to our approach for guiding the search, [13] tries to achieve a focus by learning which genetic operators to choose (which the standard IACL already does, and which is not sufficient to solve our problem of avoiding known solutions).

If we look into software testing, then the area of search-based software engineering is looking into the creation of test cases (see [1] for an overview, but also [2]). Also some works in this area look into similarity, resp. difference, between and also within solutions. As with [11, 12], these works define similarity on the structure of an individual and not on aspects of its evaluation. [14] uses a set of test cases as individuals and the difference of the elements of such a set to the other elements is measured and included in the fitness measure (essentially realizing a multi-objective optimization). Like our approach, [15] uses an already found solution, but not with the intend to avoid it but in order to use it to guide the search towards similar solutions, which is realized via the genetic operators.

Finally, with regard to using evolutionary learning for security testing or for testing wireless networks, there are no works involving previously found tests, and only a few works that are using learning techniques. In the already mentioned [16], as in other works of this group, evolutionary methods are used to find weaknesses in systems. KleeNet, [17] tries to discover bugs in wireless sensor network protocol stacks and network configurations automatically, but without involving adversaries, and consequently the misbehaviors are limited to packet loss, duplication and corruption. [18] uses evolutionary algorithms to search for network topologies that have a poor network performance when the network is subjected to automatically generated traffic. Although this work identifies such topologies, the protocols used by the network are not tested, in that no communication misbehaviours are allowed. The last two approaches might be able to make use of our improvement to IACL.

7 Conclusion and future work

We presented the idea of using similarity to known solutions in order to penalize individuals in a search run as a solution to the problem of having repeated runs of a learning-based testing system discovering the same solutions. We tested this idea with the IACL approach for testing ad-hoc wireless networks and instantiated the general method for an application in precision agriculture. Our experiments showed that we are indeed able to focus runs away from already known solutions, even in an iterated setting that adds solutions to the set of solutions to avoid over time.

Future work obviously should first evaluate our method for more applications of IACL (like the ones described in [6]) and then for other approaches to learning-based testing (like [3, 4], as mentioned before).

References

1. McMinn, P.: Search-based software test data generation: A survey, Software Testing Verification and Reliability 14, pp. 105–156. (2004)
2. Baars, A.I., Lakhotia, K., Vos, T.E.J., Wegener, J.: Search-Based Testing, the Underlying Engine of Future Internet Testing. In Proc. FedCSIS'11, pp. 917–923. IEEE Press (2011)
3. Blackadar, M., Denzinger, J.: Behavior Learning-based Testing of Starcraft Competition Entries. In Proc. AIIDE 2011, pp. 116–121. AAAI Press (2011)
4. Hudson, J., Denzinger, J., Kasinger, H., Bauer, B.: Efficiency Testing of Self-adapting Systems by Learning of Event Sequences, In Proc. ADAPTIVE-10, pp. 200–205. (2010)
5. Bergmann, K.P., Denzinger, J.: Testing of Precision Agricultural Networks for Adversary-Induced Problems. In Proc. GECCO 2013, pp. 1421–1428. IEEE Press (2013)
6. Bergmann, K.P., Denzinger, J.: Automated testing for cyber threats to ad-hoc wireless networks. In Proc. CICS 2014, pp. 34–41. IEEE Press (2014)
7. Bergmann, K.P.: Vulnerability Testing In Wireless Ad-hoc Networks Using Incremental Adaptive Corrective Learning. PhD thesis, Department of Computer Science, University of Calgary (2014) `http://theses.ucalgary.ca/bitstream/11023/1504/4/ucalgary_2014_bergmann_karel.pdf`
8. Postel, J.: Internet Protocol, RFC 791 (Standard) (1981)
9. Postel, J.: User Datagram Protocol, RFC 768 (Standard) (1980)
10. Perkins, C., Belding-Royer, E., Das, S.: Ad hoc On-Demand Distance Vector (AODV) Routing, RFC 3561 (Experimental) (2003)
11. Denzinger, J., Kidney, J.: Improving Migration by Diversity. In Proc. CEC 2003, pp. 700–707. (2003)
12. de Jong, E.D., Watson, R.A., Pollack, J.B.: Reducing Bloat and Promoting Diversity using Multi-Objective Methods. In Proc. GECCO-01, pp. 11–18. (2001)
13. Maturana, J., Saubion, F.: A Compass to Guide Genetic Algorithms. In Intl. Conference on Parallel Solving from Nature, pp. 256–265. Springer (2008)
14. Bueno, P.M.S., Jino, M., Wong, W.E.: Diversity Oriented Test Data Generation using Metaheuristic Search Techniques, Information Sciences 259, pp. 490–509. (2014)
15. Yoo, S., Harman, M.: Test Data Regeneration: Generating New Test Data from Existing Test Data, Journal of Software Testing, Verification and Reliability 22(3), pp. 171–201. (2012)
16. Kayacik, H.G., Zincir-Heywood, A.N., M.I. Heywood: Can a good offense be a good defense? Vulnerability testing of anomaly detectors through an artificial arms race, Soft Comput. 11(7), pp. 4366–4383. (2011)
17. Sasnauskas, R., Landsiedel, O., Alizai, M.H., Weise, C., Kowalewski, S., Wehrle K,: Kleenet: discovering insidious interaction bugs in wireless sensor networks before deployment, In Proc. IPSN'10, pp 186–196. (2010)
18. Woehrle, M.: Search-based stress testing of wireless network protocol stacks, In Proc. ICST'12, pp 794–803. (2012)