Evaluating Goal Ordering Structures for Testing Harbour Security Policies

Chris Thornton, Tom Flanagan, Jörg Denzinger, Jeffrey E. Boyd
Department of Computer Science, University of Calgary
Calgary, Canada
Email: {chris.thornton,thomas.m.flanagan,denzinge,jboyd}@ucalgary.ca

***Abstract*—Large, complex systems can exhibit unforeseen behaviours. In the case of surveillance and security systems, these behaviours can be weaknesses that should be discovered by automated testing and ameliorated. Previous work has shown that such automated testing can be done using particle swarm optimization to learn behaviours that allow a set of attackers to defeat the system. However, for the optimization to succeed, it must have some knowledge about what constitutes a successful attack in order to guide the swarm. This knowledge is encapsulated in a goal ordering structure. In this paper, we examine the goal ordering structure and its role in the learning of system weakness. We specifically look at applications in harbour surveillance and security, and show how knowledge of the likely properties of a successful attack can be added to the goal ordering structure. Our experimental results show that adding knowledge to the goal ordering structure improves the search, when that knowledge is correctly inserted into the structure.**

***Index Terms*—testing MAS, particle swarm optimization, unwanted behavior, MAS simulation**

I. Introduction

Surveillance and security systems capable of observing and protecting large installations like a harbour or an international border are large and complex. The size of the system is an inevitable consequence of the size of facility being protected, but the complexity arises from several factors including:

- variety in sensors and platforms,
- the dynamic nature of sensors and platforms,
- the dynamic nature of the operating environment, and
- deployment policies.

A variety of sensors (such as visual, infrared, sonar, and radar) is necessary as each has strengths and weaknesses that vary over operating conditions. Platforms that cary the sensors are often dynamic, e.g., pan-tilt (PT) heads change the field of view of a camera, and mobile platforms (both manned and unmanned) change a sensor's position. The operating environment is also dynamic as weather and human activity (e.g., legitimate traffic in a harbour or workers in a secure facility) changes. Finally, sensors and their platforms all have some underlying deployment policy. E.g., a camera on a PT head has a policy to determine where it should aim and when, and a mobile sensor platform has a policy to tell it where it should be located and when, how it should get there, and how it should respond to events that occur.

As with any complex system of this size, it is difficult to predict how it will behave in all circumstances. In some cases, these behaviours are unwanted vulnerabilities and it is desirable to discover them and ameliorate their impact on the success of the system. Multi-agent simulations have proven to be useful tools for human decision makers to evaluate complex systems (e.g., see [1] and [2]). By controlling some agents in a simulation, a tester can easily test a system for the desired behaviours. However, it is difficult for a human to test for unpredicted behaviours, and automated support for testing is a necessity.

[3] presents an approach to provide this automated support. The method learns coordinated behaviours for a set of *attack* agents that result in the defeat of the subject system, i.e., a success in learning means a system vulnerability is found. The space of possible attack agent actions is large, so particle swarm optimization (PSO) is used to learn the behaviors. A *goal ordering structure* guides the swarm towards successful attacks - the *goal ordering structure encapsulates knowledge about what is likely to make a successful attack*. Although [3] defines the goal ordering structures and shows one example, other goal ordering structures are possible, and effective use of other goal ordering structures can improve the learner's ability to find a weakness in the subject system.

In this paper, we show how additional knowledge can be incorporated into a goal ordering structure to improve the learning of attacks. We consider the testing of harbour surveillance and security systems, and including knowledge specific to attacks in such an application. To demonstrate, we add to the goal ordering structure the knowledge that a successful attacker should avoid defending patrol vessels. This is not trivial - done naively, such knowledge can lead the learner to favour behaviours in which the attackers never enter the harbour. Done properly, the additional knowledge in the goal ordering structure helps the learner to find attacks more effectively, and to find qualitatively different attacks. We then postulate expectations for variations of the improved goal ordering structure and experimentally verify these expectations. Our results show that goal ordering structures are an effective way to guide search, provided that the knowledge is inserted into the structure in a suitable place.

II. Basic concepts

This section introduces our method of testing by learning, and then provides a short introduction to particle swarm optimization.

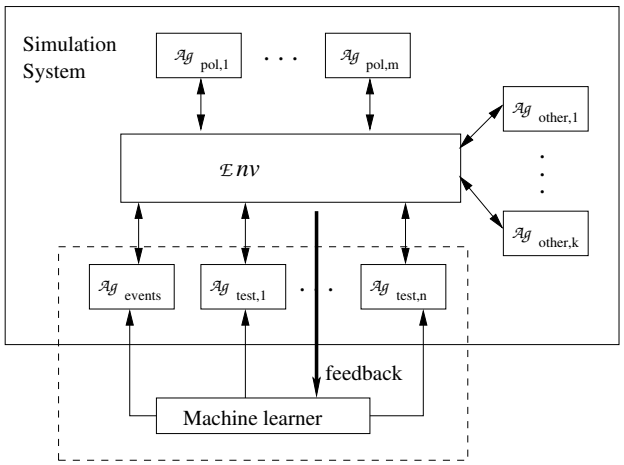

Fig. 1. A block diagram of our concept for testing policies by learning cooperative behavior. The machine learner learns behaviours for the *test* agents coordinated with the *events* agent, to defeat the *policy* agents in the simulated environment.

A. Testing policies by learning behavior

Figure 1 shows our concept for learning behaviours to test a subject system. All testing occurs in a simulated environment, $\mathcal{E}nv$ within which agents interact. Our subject system (the system being tested) consists of the policy agents A_{pol} = $\{\mathcal{A}g_{pol,1},...,\mathcal{A}g_{pol,m}\}$. While we refer to these as policy agents, we are in fact testing all properties of these agents that are simulated. A set of other agents, A_{other} = $\{\mathcal{A}g_{other,1},...,\mathcal{A}g_{other,k}\}$ includes agents that are in the environment, but are not the subject of testing. For example, in harbour surveillance, other agents would include legitimate harbour traffic. The set of test agents, A_{test} = $\{\mathcal{A}g_{test,1},...,\mathcal{A}g_{test,n}\}$, are the attackers that test A_{pol}. Finally, the system includes an events agent, $\mathcal{A}g_{events}$, that can trigger environmental changes in the simulation. Physically, this could correspond to changes in weather or daylight.

In this context, testing is done by learning behaviors for A_{test} and $\mathcal{A}g_{events}$ that defeat A_{pol}, thereby revealing weakness. While learning behaviours could be done by a human, we use a machine learner to automate the process. Note that while agents in A_{test} do not control environmental events, they can exploit them in attacks. Therefore, the learner must control both A_{test} and $\mathcal{A}g_{events}$.

Let $Act_{test,i}$ be the set of possible actions that an agent $\mathcal{A}g_{test,i}$ can perform, and Act_{events} be the set of events that $\mathcal{A}g_{events}$ can invoke in the simulation environment. Then the learning process works as follows.

1) The learner selects a sequence of actions for A_{test} and $\mathcal{A}g_{events}$ from $Act_{test,i}$ and Act_{events}.
2) The environment simulation runs, producing a series of observations of the environment, $e_0, e_1, \ldots, e_s$.
3) Based on an evaluation of the observations, the learner selects a new sequence of actions to test.

Our learner uses particle swarm optimization to steer the selection of actions towards successful attacks, using the goal ordering structure to compare the observations. As such, step 2 does several simulation runs per iteration of the system – one simulation run per particle in the swarm.

B. Particle Swarm Optimization

Particle Swarm Optimization (PSO) [4] emulates moving particles with the attraction behaviour of members of a swarm, to perform a search in a solution space. The search state is a set of *particles* p_i, $i = 1, 2, \ldots, l$, each of which is characterized by its current position, pos_i, its current velocity, v_i, and its best position $best_i$ in the past. (Note that this is a velocity in the solution space of an optimization problem, and not to be confused with the velocity of physical agents in our simulation.) The position of a particle is usually a vector of continuous variables that represent a solution to the instance of the search problem. In the basic case there is a single *goal function*, f, defining the quality of a position (i.e., what is best) and the goal of the search is to find a position that is optimized with respect to f.

The search in the basic case is performed by updating each particle in the state according to the following equations:

$$v_i^{new} = Wv_i + C_1r_1(best_i - pos_i) + C_2r_2(Best - pos_i), \quad (1)$$

$$pos_i^{new} = pos_i + v_i^{new}, \quad (2)$$

where W is a weight parameter controlling the influence of the previous velocity, C_1 is the *cognitive learning factor*, C_2 the *social learning factor* and $r_1, r_2 \in [0, 1]$ change randomly during the search. $Best$ is the best position the whole swarm has found up to any iteration of the search. Iterations terminates either after a given number of update rounds or when $Best$ fulfills certain conditions. $Best$ is the *optimum* solution output by the algorithm.

For many applications, there is no single goal function describing what is searched for, but instead we have a vector $\vec{f}$ = $(f_1,...,f_q)$ of goal functions, leading to multi-objective optimization. In these cases, we are not interested in a single solution, since there is not one position that is optimal for all goal functions, and positions that are very good for one f_i are often not as good for an f_j $(i \neq j)$. PSO can be easily extended to deal with multi-objective optimization (see [5] for an overview).

A key concept of multi-objective optimization (and for our goal ordering structures) is the *domination* of one solution, pos_1, over another solution, pos_2, denoted by $pos_1 \succ_{\vec{f}} pos_2$ which is defined by $f_i(pos_1) \geq f_i(pos_2)$ for all i (if our goal is to maximize all functions in $\vec{f}$). The subset PF of all possible solutions Sol to a multi-objective optimization problem where for each $x_1 \in PF$ we have that there is no $x_2 \in Sol$, $x_1 \neq x_2$, such that x_2 dominates x_1 is the so-called Pareto-front of the particular instance of the problem.

In this work, we use a variant of PSO that considers dominance among goal functions by extending the definition of a particle to a triple p_i = $(pos_i,v_i,Ownbest_i)$. The set $Ownbest_i$ records all previous positions of p_i that are not dominated by any of the other previous positions of p_i. Instead

of just one solution, $Best$, for the whole particle swarm, we replace $Best$ in Equation (1) with a value selected from the sets $Ownbest_{(i-1) \ mod \ l}$ and $Ownbest_{(i+1) \ mod \ l}$ of non-dominated solutions of the "neighbours" of particle p_i. The selection is done randomly every time Equation (1) is applied, as is the selection of an element from $Ownbest_i$ to play the role of $best_i$ in Equation (1). After the new position of p_i is created, it is checked for domination by an element of $Ownbest_i$. If it is not, it is added to $Ownbest_i$ and all elements in $Ownbest_i$ that are dominated by pos_i^{new} are removed from it. Again, the search terminates when it reaches a maximum number of iterations, or the union of all $Ownbest_i$s fulfills certain conditions.

III. Testing of harbour security policies

This section describes how we combine the testing concept described in Section II-A with PSO in Section II-B to test deployment policies for agents in a harbour surveillance and security system.

A. Harbour simulations for security policies

Harbours are an important part of a country's infrastructure that need to be protected. Part of this protection is having harbour defenders detect and intercept suspicious vessels. Due to resource limitations, these defenders need to be well guided and coordinated, which is usually achieved by creating a good policy for them. One aspect of a good policy is to stop attacks before they reach their goals. This requires that defenders detect any possible threat to installations or ships, investigate a possible threat, and if the threat is real, neutralize the threat.

Using our notations from Section II-A, the defender vessels are the policy agents, A_{pol}. The essential features of these agents needed for simulation are their sensor capabilities and movement capabilities. Legitimate users of a harbour form A_{other} and the test agents, A_{test} are the attackers. We say that an attack has succeeded when a test agent reaches a defined position in the harbour without being intercepted.

The environment, $\mathcal{E}nv$, at the centre of our simulation system uses Geographic Information System (GIS) technology [6]) enhanced to simulate movement and sensor-based perception of all agents. The necessary geographical data for the GIS for our experiments came from the Government of Canada's National Topographic Data Base which is available from GeoGratis [7]. Given our application where the agents are often constantly moving, we had to decide on how to update the positions of the agents in the GIS. In our experiments, the movement of all agents is computed in frames of $0.1s$ using Euler integration on forces acting on the vessel. These forces are boat drag, throttle and the rudder positions as provided by the vessel. This means that all of these updates are available as environment states to the learner.

B. Using PSO to test harbour security policies

To apply our testing scheme from Section II-A to test harbour security policies, we need an agent architecture for the agents in A_{test} and a machine learner that can learn their behaviors. The set of actions available to the test agents is limited to moving around the harbour at variable speeds. Variation in speed is necessary for attacks to exploit timing. Therefore, the test agent architecture works as follows.

- We use a small number of high-level waypoint-speed pairs, $((x, y), speed)$, to define the trajectory of an attacker. This reduces the dimensionality of the search space.
- Obstacles such as land or hazards to navigation may obstruct a straight path between waypoints, so the agents use a path planner (we used [8]) to establish low level waypoints on a short, safe path between high-level waypoints.
- Speed is a normalized throttle setting in the range $[0.1 \ldots 1]$.
- Waypoints that fall on land are moved to the nearest navigable position on the water.

So for a harbour attacker, a PSO particle position is a sequence of waypoints with speeds for each agent, given by

$$(((x_{1,1}, y_{1,1}, speed_{1,1}), \ldots, (x_{1,l_1}, y_{1,l_1}, speed_{1,l_1})), \ldots, ((x_{n,1}, y_{n,1}, speed_{n,1}), \ldots, (x_{n,l_n}, y_{n,l_n}, speed_{n,l_n}))),$$

as illustrated in Figure 2. The learner evaluates a particle with the observations obtained from a simulation run in which attackers use the waypoints and speeds defined within the

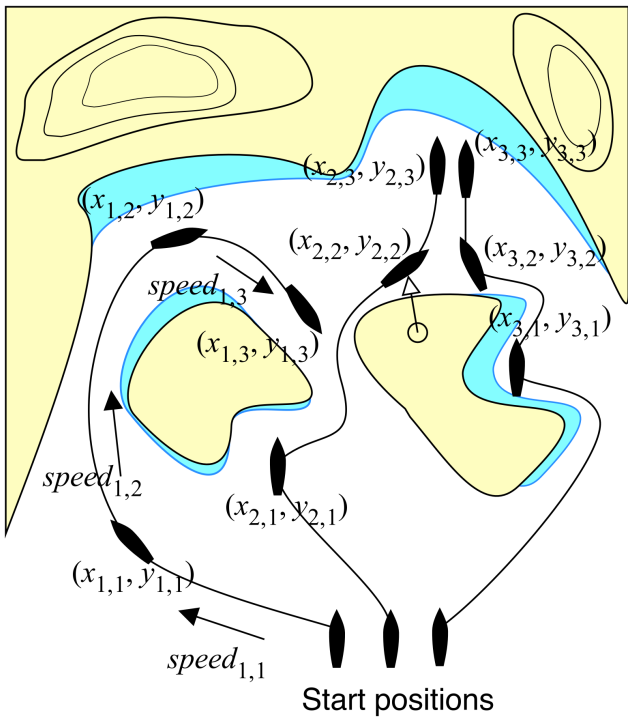

Fig. 2. Contents of a particle position in our swarm optimization learning system plotted on a caricature nautical chart (yellow is land, blue is shallow water), and white is deep water. A particle is a set of waypoints and speeds for n agents (here $n = 3$, speeds shown only for the first agent). The actual path between waypoints is found by the method in [8]. When a waypoint falls on land (as seen on the island on the right), the simulation moves the waypoint to the nearest point of water.

particle position. In our current version, A_{other} is the empty set and we are also not creating any events, so that there is no $\mathcal{A}g_{events}$.

There are several measures that the learner takes from each particle position's simulation run in order to compare particle positions. Since this is at the centre of our goal ordering structures we will look more closely at this part of our PSO in the next section.

Since our particles are vectors of numbers, they can be updated as described in Section II-B. The initial positions for our particles are created using random values between 0.1 and 1 for all speeds needed and while each waypoint is also randomly chosen, we limited the randomness by requiring that each waypoint is less than 600*m* away from its predecessor in the waypoint sequence for an attack agent. The agents in A_{test} start outside the harbour at given coordinates that are the same in each simulation run. The simulation run for a particle position ends when

- the attack objective is fulfilled (i.e., an attacker has reached the target position),
- all attackers have been intercepted (i.e., the A_{pol} agents were successful), or
- all attackers are at the end of their waypoint sequences.

IV. Goal ordering structures for breaking harbour security policies

There are many possible things to measure in a simulation run of a particle position as described in the last subsection. Naturally, a particle position that results in one attacker reaching its target reveals a weakness and fulfills the goal of the learning process. But such a goal function is *all-or-nothing* - if the target is not met, it gives no indication of where to search for better solutions. As such, we need other knowledge about the characteristics of good attacks to steer the search.

This knowledge comes in the form of additional goal functions based on measurements from simulation runs that indicate how near the behaviors of the agents come to achieving their goal, or in the case of PSO, we need to compare two positions to determine which is closer to achieving the goal. There are many possible, reasonable goal functions but none alone catches the whole picture, and just summing up various measures (even a weighted sum) runs into the problem that some measures are contradicting each other (see below).

To deal with these problems, [3] introduced the concept of a *goal ordering structure* that creates a hierarchy of goal functions. Each hierarchy level represents a set of measures that are treated like the objectives of a multi-objective optimization, and the different levels then represent a lexicographic combination of orderings. Our contribution in this paper is to move beyond the single example of a goal ordering structure in [3] to include more knowledge in the goal ordering structure to steer the PSO and thereby improve the effectiveness of testing.

A. Goal Ordering Structures

Formally, the idea of a goal ordering structure ,$\rhd$, is as follows. For two particle positions, pos_1 and pos_2, a goal ordering structure has the form

$$(\{f_{11}, ..., f_{1q_1}\}, \ldots, \{f_{u1}, ..., f_{uq_u}\}),$$

(or $(\vec{f_1}, \ldots, \vec{f_u})$ for short), where f_{ij} is a goal function that assigns a measure to a particle position by evaluating the observations, $e_0, \ldots, e_s$ that result from simulation of the waypoints and speeds in that position. If $\rhd$ denotes this ordering structure, then we have

$$pos_1 \rhd pos_2,$$

if

$$\begin{array}{lll} pos_1 & \succ_{\vec{f_1}} & pos_2, \text{ or} \\ pos_1 & =_{\vec{f_1}} & pos_2 \text{ and } pos_1 \succ_{\vec{f_2}} pos_2, \text{ or } \ldots \text{ or} \\ pos_1 & =_{\vec{f_1},\ldots,\vec{f_{u-1}}} & pos_2 \text{ and } pos_1 \succ_{\vec{f_u}} pos_2. \end{array}$$

In this context, $pos_1 =_{\vec{f_i}} pos_2$ means that pos_1 and pos_2 have an identical value in each of the measures in $\vec{f_i}$, and $=_{\vec{f_1},\ldots,\vec{f_i}}$ means $=_{\vec{f_1}}$ and $=_{\vec{f_2}}$ and ... and $=_{\vec{f_i}}$. As such, $\rhd$ represents a lexicographical combination of multi-objective domination orderings, which due to the partiality of the domination orderings, is itself a partial ordering. Consequently, it is possible that two positions might not be comparable, and we must use a multi-objective version of PSO, even though we have a single ultimate goal.

B. Goal functions for harbour security

Proper selection of goal functions within a goal ordering structure encapsulates knowledge and intuition about an application domain to steer PSO. The following five measures encapsulate our knowledge and intuition of harbour surveillance and security. Let $\mathbf{e} = (e_0, \ldots, e_s)$ be a trace of measurements obtained from the simulation of a particle position pos.

1) Interception: The *intercept* measure considers the number of attackers that are intercepted by the defenders (fewer interceptions is better):

$$f_{intercept}(\mathbf{e}, pos) = \begin{cases} 0, & \text{if there is a } j, \text{ such} \\ & \text{that all } \mathcal{A}g_{test,i} \\ & \text{are intercepted in } e_j \\ 1, & \text{else} \end{cases}$$

with $pos_1 \succ_{intercept} pos_2$, if

$$f_{intercept}(\mathbf{e}, pos_1) > f_{intercept}(\mathbf{e}, pos_2).$$

2) Success: The *success* measure evaluates weather or not the attack was successful:

$$f_{success}(\mathbf{e}, pos) = \begin{cases} 1, & \text{if there are } j, i, \text{ such} \\ & \text{that } \mathcal{A}g_{test,i} \text{ reached} \\ & \text{the target spot in } e_j \\ 0, & \text{else} \end{cases}$$

with $pos_1 \succ_{success} pos_2$, if

$$f_{success}(\mathbf{e}, pos_1) > f_{success}(\mathbf{e}, pos_2).$$

3) Distance to target: The *dist* measure indicates how near an attacker came to the target, at any time:

$$f_{dist,i}(\mathbf{e}, pos) = \sum_{j=1}^{\lfloor s/100 \rfloor} dist(e_{100j}, \mathcal{A}g_{test,i}) + dist(e_s), \mathcal{A}g_{test,i}) \quad (3)$$

where $dist(e, \mathcal{A}g_{test,i})$ is the length of the shortest path created from the position of $\mathcal{A}g_{test,i}$ in e to the target (again computed using path finding). The factor of 100 in Equation (3) corresponds to 10*s* for a 0.1*s* measurement sample period in the simulation. We define $pos_1 \succ_{dist,i} pos_2$, if

$$f_{dist,i}(\mathbf{e}, pos_1) < f_{dist,i}(\mathbf{e}, pos_2).$$

4) Avoid defenders: The *hide* measure reflects the idea that attack agents should avoid the patrolling agents, and therefore measures the distance of a particular attacker to the defenders, at any time:

$$f_{hide,i}(\mathbf{e}, pos) = \sum_{j=1}^{\lfloor s/100 \rfloor} ndist(e_{100j}, \mathcal{A}g_{test,i}) + ndist(e_s, \mathcal{A}g_{test,i}), \quad (4)$$

where $ndist(e, \mathcal{A}g_{test,i})$ is the shortest distance between $\mathcal{A}g_{test,i}$ and any of the vessels in A_{pol} in e. The factor of 100 in Equation (4) corresponds to 10*s* for a 0.1*s* measurement sample period in the simulation. We define $pos_1 \succ_{hide,i} pos_2$, if

$$f_{hide,i}(\mathbf{e}, pos_1) > f_{hide,i}(\mathbf{e}, pos_2).$$

5) Group avoidance of defenders: The *hidesum* measure is a summation of the individual *hide* measures over all attackers, and as such indicates how the attackers as a group are avoiding defenders:

$$f_{hidesum}(\mathbf{e}, pos) = \sum_{i=1}^{n} f_{hide,i}(\mathbf{e}, pos),$$

where $pos_1 \succ_{hidesum} pos_2$, if

$$f_{hidesum}(\mathbf{e}, pos_1) > f_{hidesum}(\mathbf{e}, pos_2).$$

C. Assembling goal ordering structures

We now look at the task of assembling a goal ordering structure from a collection of goal functions. As a starting point, consider the goal ordering structure used in [3], $\rhd_{base}$, given by:

$$(\{f_{intercept}\}, \{f_{dist,1}, ..., f_{dist,n}\}, \{f_{success}\}).$$

But with more knowledge and intuition (in the form of goal functions) where do we insert additional goal functions to help the search? Here, helping can mean accelerating learning (finding more successful attacks in a shorter time), or finding attacks that might not otherwise be found without the additional knowledge.

If we look at $\rhd_{base}$, then placing the *hide* and *hidesum* measures before or with the *intercept* component or after or with the *success* component does not make a lot of sense – the former leads to attackers staying out of the harbour and the latter has no net effect. The remaining options are: before the *dist* component, in the *dist* component, or after the *dist* component. Putting them into the *dist* component leads to contradicting measures – simultaneously approaching the target while staying away from defenders. Therefore we consider the following new goal ordering structures. $\rhd_{hidebefore}$ is

$$(\{f_{intercept}\}, \{f_{hide,1}, ..., f_{hide,n}\}, \{f_{dist,1}, ..., f_{dist,n}\}, \{f_{success}\}),$$

and $\rhd_{hideafter}$ is

$$(\{f_{intercept}\}, \{f_{dist,1}, ..., f_{dist,n}\}, \{f_{hide,1}, ..., f_{hide,n}\}, \{f_{success}\}).$$

Similarly, $\rhd_{hsumbefore}$ is

$$(\{f_{intercept}\}, \{f_{hidesum}\}, \{f_{dist,1}, ..., f_{dist,n}\}, \{f_{success}\}),$$

and $\rhd_{hsumafter}$ is

$$(\{f_{intercept}\}, \{f_{dist,1}, ..., f_{dist,n}\}, \{f_{hidesum}\}, \{f_{success}\}).$$

It is our expectation that the ordering structures where we put the new measures before the *dist* component will not be successful, since they will allow the learner to keep generating attack strategies where the attackers stay away from the harbour. In contrast, we expect the ordering structures where the new measures are behind the *dist* component will improve upon $\rhd_{base}$.

V. Experimental evaluation

In order to present our experimental evaluation of the different goal ordering structures, we first describe the different policies and harbours that we use in our evaluation and the general set-up of the simulation and testing systems. Then we present and comment on the quantitative data from our experiments to verify our expectations for the different ordering structures. Finally, we take a closer look at some of the attack strategies our testing system finds with the different ordering structures.

A. Experimental set-up

In our experiments, we use the two harbour patrol and interception policies from [3]. They are both high-level policies suitable for any harbour and any target location.

The first policy, *pat-int*, divides the agents in A_{pol} into two subtypes: namely patrollers and interceptors. The patrollers detect potential attackers and alert the interceptors. In turn, the interceptors approach the intruder, identify it and, if necessary, destroy it. Upon detection, the patroller provides the interceptor with the course of the potential intruder as long as it is in its sensor range. If an intruder comes close enough to a patroller to be identified, then the patroller will destroy it, otherwise patrollers stay to a predetermined patrol circuit. The interceptors wait at predefined positions in the harbour and only become active when alerted by a patroller. When active, an interceptor determines the best course to approach an intruder, based on the information from the patroller. If the

interceptor fails to find an intruder at the expected position then the interceptor returns to its waiting position.

Our second policy, $all-pat$, does not partition the agents in A_{pol}. All patrollers follow a circuit around the harbour and the available vessels are evenly spaced on this circuit. If a patroller detects a potential intruder, the available agent closest to the intruder is sent to identify the boat and if this identification results in the need to intercept, then this agent will intercept the intruder.

In our simulation system, we have two possible environments: Esquimalt harbour on Vancouver Island in Western Canada, and Halifax harbour in Eastern Canada. The instantiations of the general policies were hand-coded by us and communication between the agents in A_{pol} was achieved using the GIS. This means that there were no communication failures possible. For $pat-int$ and Esquimalt this meant having two patrollers and two interceptors. One patroller circles the mouth of the harbour, while the second patroller, the "goaltender", does a small circle very close to the target. The two interceptors have their idle positions near the dock adjacent to the target spot. The target is placed deep inside the harbour behind a pier. Policy $pat-int$ for Halifax also uses two patrollers and two interceptors, with the same idea for the patrollers, i.e., one circling the mouth of the harbour and one doing its patrol route relatively near to the target. For policy $all-pat$ in Esquimalt we used four patrollers circling inside the harbour. We also used four patrollers for the Halifax scenarios. In all scenarios, the sensor perception by an $\mathcal{A}g_{pol,i}$ was modelled as a circular region about the agent with a radius of 300 meters, within which the agent has perfect sensing. To determine whether an agent is a threat or not, an $\mathcal{A}g_{pol,i}$ needs to get within 20 meters of this agent. Note that this is a simplification of typical physical sensors that favours the defenders – a more realistic sensor model would be easier for the attackers.

The two policies and their instantiations for the two harbours have weaknesses, even if we limit the number of agents in A_{test}. The limited experiments in [3] showed weaknesses with regard to being able to sneak by all agents in A_{pol} for both policies and with regard to using some of the agents in A_{test} as decoys, drawing the defenders out of position thereby giving another attacker free access to the target. We consider attacks where the defenders fail to detect any of the intruders to be the most dangerous weakness (other attacks are easily ameliorated). Therefore, part of our goal in looking at new ordering structures is to find more successful strategies that avoid detection.

In our experiments, we set the parameters of our testing system as follows: the PSO parameters are W = 0.8, C_1 = 0.2, and C_2 = 0.4. The number of waypoints for an $\mathcal{A}g_{test,i}$ in an attack strategy is 10 and we use 20 particles. We have a maximum of 100 position updates per particle and every entry in Table I is based on performing at least 100 iterations of the testing system (due to the random factors involved in the PSO, repeating trials, without special measures, results in different outcomes). Due to the heterogeneity in the machines used for doing the experiments we use the average number of particle updates (generation) as speed measure.

B. Quantitative analysis

Table I reports our quantitative results. For the success rates of our system, if we look at $\rhd_{base}$, we see that the $all-pat$ policy for Esquimalt presented more of a challenge for our testing system than the other scenarios, but still, nearly half of the testing runs were successful, so that the basic goal ordering structure is already good at helping to reveal weaknesses in the policies. But the ordering structures that put the new measure in a level after the distance-to-target measures were able to improve the success rates in half of the scenarios, with the highlight being an improvement of more than 10 percent for $\rhd_{hideafter}$ for the two agent attack of the $all-pat$ policy for Halifax. For the scenarios with no improvements, the success rates were only slightly worse, and the average number of particle updates to find a weakness was usually less than for $\rhd_{base}$. The worst difference in success rate was 3%. As expected, having either of the new measures before the distance-to-target measures was not successful, although we were surprised that there were so many successful runs and that those successful runs usually needed fewer particle updates than the other ordering structures. We will look into this in more detail in the next subsection. The average number of particle updates until a successful attack, found by averaging over successful runs of the testing system, in general favours $\rhd_{hideafter}$ and $\rhd_{hsumafter}$ over $\rhd_{base}$. So, with regard to putting the new measure after the distance-to-target level, we conclude that there are some improvements.

With regard to combining the measures of the individual agents or having them represented individually, our initial expectation was that the $f_{hidesum}$-measure would perform clearly better than using the $f_{hide,i}$-measures in a multi-objective fashion. Our rationale was that while $f_{hidesum}$ allows for tradeoffs, (e.g., one agents gets nearer to an $\mathcal{A}g_{pol,i}$ near the target, but another agents gets farther away from all agents in A_{pol}), the not-dominated requirement for the $f_{hide,i}$-measures would not allow for tradeoffs. Instead many particle positions where one attacker gets nearer to an $\mathcal{A}g_{pol,i}$ near the target would be dominated by positions where this particular attacker stays farther away. As the table shows, with regard to success rate, the $f_{hidesum}$-variant is better than the associated $f_{hide,i}$-variant in nine of 16 cases, versus $f_{hide,i}$ is better six times (with 1 tie). If we look at the average generation of success, in 10 cases $f_{hidesum}$ is faster than $f_{hide,i}$, with five cases the other way around (again with 1 tie). As we will demonstrate in the next subsection, the reason for this is that we can also achieve some tradeoffs when using $f_{hide,i}$ (although it depends on what initial positions are created for the particles), which explains the unexpected performance of the $f_{hide,i}$s.

C. Selected attack strategies

Due to lack of space, we cannot provide detailed information on individual runs of our testing system using the

	success rates (in percent)								average successful generation							
Harbour:	Esquimalt				Halifax				Esquimalt				Halifax			
Policy:	$pat-int$		$all-pat$		$pat-int$		$all-pat$		$pat-int$		$all-pat$		$pat-int$		$all-pat$	
Agent numbers:	2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3
$\triangleright_{base}$	67.3	74.5	46.6	57.1	77.2	84.2	68.1	92.1	17.2	16.2	16.6	15.6	23.2	21.0	18.4	16.5
$\triangleright_{hideafter}$	64.0	75.5	45.7	57.7	75.5	89.6	79.5	85.4	15.8	17.5	15.9	15.1	23.7	20.7	18.2	16.4
$\triangleright_{hsumafter}$	67.3	77.6	45.7	58.2	73.3	88.4	75.5	88.5	16.9	16.4	14.9	14.2	22.8	19.6	18.2	16.8
$\triangleright_{hidebefore}$	23.6	38.6	13.6	28.6	44.9	68.3	66.7	82.6	13.2	16.9	10.9	14.5	25.5	22.1	19.1	14.6
$\triangleright_{hsumbefore}$	9.6	34.2	15.2	19.7	53.4	68.4	68.8	89.2	18.1	14.5	15.1	11.5	23.5	22.6	15.8	13.9

TABLE I
COMPARISONS BETWEEN GOAL ORDERING STRUCTURES

different goal ordering structures. In particular, we cannot provide any screenshot sequences demonstrating interesting observed behaviors in the successful attacks. However, [9] provides such information. See also *http://www.cpsc.ucalgary.ca/-∼denzinge/papers/Movies/harbour/overview.html*. In the following, we report on generally observed trends.

If we look at the differences between the attack strategies that the different ordering structures produced in our testing system for the same problem scenario, we see that the particular ordering structure clearly influences the "ideas" behind the attacks. While the ordering structures that put the new measures after the distance-to-target measures essentially use one attacker in a timed attack, and have the other attacker stay outside of the harbour near its starting point, the ordering structures that give priority to staying away from detection send one attacker far away from the harbour to counterbalance the need for the other attacker to come near the defenders in order to achieve the ultimate goal. In order for such a behavior to evolve, it is important to have among the initial positions a representation of this general pattern (i.e., one attacker has entered the harbour and has come near to the defenders while the other attacker counters this) that is not dominated by other positions, which explains the relatively low success rates. However, since such a pattern is also not uncommon, and therefore likely to occur in randomly created particle positions, it also explains why we have success from time to time.

Another surprising result from our result tables was that there was not much difference between the $f_{hidesum}$- and $f_{hide,i}$-variants of our goal ordering structure. $f_{hidesum}$ allows for counterbalancing between the attackers, but we did not think that using $f_{hide,i}$ would. But our testing system found a way for this, namely having the attacker that will later be the successful one doing what we call "accumulating hiding credit". This can be in form of a little bit of a loop or zig-zag in approaching the harbour creating more environment states where it stays far away from defenders before getting near them (which is done rather directly and quickly).

All the attack behaviors we looked at show clearly how much influence the goal ordering structure has on what attack strategies will be developed and that it is not so easy to predict exactly what the outcome of the learning will be, at least with regard to details. But they also show that the basic learning method is rather robust, able to overcome "unuseful" advice by a goal ordering structure, which is very important for all kinds of testing. Also, the found strategies are not exactly along the lines a human would test - we are going beyond what human testing can do.

There were also a few test runs that produced solutions that highlight features of our testing approach and that revealed some unexpected problems in our implementation of the defenders. In one of these, an attacker has a waypoint that is just shy of the shoreline, and since it was travelling at full throttle, it was not able to turn fast enough to avoid collision with the land. Despite this, the other attacker is still able to find the right timing to slip between two patrollers to reach the target area. The crash is the interesting part of this attack strategy, because it shows that our learner naturally is not aware of the laws of physics and consequently the simulator needs to uphold them. Usually, crashing attackers is not good, so that just based on the feedback the learner will avoid such strategies, but if there is still success possible, it does not care. While we do not have other types of defenders (or emergency personnel and emergency policies) in our simulation, creating an emergency would be a good way to draw attention away from the real attack and our learner obviously can do so (without even knowing what emergencies are).

In another successful test run, one interceptor collides with a patroller (since we forgot to implement collision avoidance between the defenders) and while the general policy allowed for dealing with this (by having the other interceptor taking on the role of a patroller), due to this an attacker can make it to the target without being intercepted, even though it is spotted by the interceptor turned patroller now following patroller behavior. Our goal ordering structures clearly were not aiming at testing our policies for errors like this one, but it was nevertheless detected (although not in many of our testing runs, obviously). But we consider this as a good example of the abilities of our approach and especially of the learner that was able to take advantage of the implementation problem to fulfill its ultimate goal.

VI. RELATED WORK

The use of search methods to test systems has, over the last years, seen growing popularity. [10] provides an overview of this area and SEBASE [11] is a large, although incomplete,

collection of papers around search-based Software Engineering, including a lot of papers on testing. Unfortunately, the question how to compare solutions has not seen a lot of attention, despite the fact that it is very important to guide the search for finding unwanted policy behavior (which also has not drawn a lot of attention in this area, with the exception of [12] and [13]).

There have been some works on creating fitness functions for evolutionary algorithms, although most looked at improving the behavior of the algorithms in general (e.g., [14] tried to keep the solutions diverse). A notable exception is [15], that looked into features of different environments for evolving wanted behaviors, but which still had to combine those features into a single measure.

Widening the comparison to any works in literature, our work, at first glance, seems to contradict the observations by [16], which is the only paper that has suggested the use of learning to create wanted behaviors within multi-agent simulations we were able to find. The main point in their observations was a brittleness in the learning results, whereas our experiments showed a robustness of our approach. But [16] looked at learning a policy, which has to deal with many possible "attacks" and therefore is a much more complex task than our learning of attacks.

VII. Conclusion and future work

We investigated different goal ordering structures for PSO-based learning of cooperative behaviors for testing harbour security policies using multi-agent simulations. By learning sequences of high-level waypoints in a spatial simulation for a group of test agents and adding low-level waypoints using a conventional path planner to create physically possible behaviors for these test agents, our method tries to get policy agents that implement the tested policies to show an unwanted behavior that reveals a weakness in the tested policy. Goal ordering structures are used to guide the learning and allow for a rather explicit representation for measurements of interesting aspects of runs of the underlying simulation system. This provides a user of such an automated policy testing system with a high-level way to guide the system.

Our experiments with testing harbour security policies showed that goal ordering structures indeed allow a good guidance of the learning, mostly achieving the predicted effects, but also that there is still potential for surprises (although most of them were positive). The experiments also provided a few positive side effects, like revealing an error in our implementation of the policies. Also, many of the learned behaviors that revealed weaknesses are rather unusual compared to obvious test cases that humans would create, so that the automated test system represents at least a good additional testing tool.

Our future work will look into applying our method to other application areas that use spatial simulations. Also, extensions of our current system, like adding other harbour users (we are currently collecting data from the harbours to be able to model some of these users), weather events or more sophisticated sensor models, and how these extensions have to be incorporated in the testing problem are part of our future plans.

References

[1] D. Power and R. Sharda, "Model-driven dss: concepts and research directions," *Decision Support Systems*, vol. 43, no. 3, pp. 1044–1061, 2007.

[2] A. Baldwin, M. Mont, and S. Shiu, "Using modelling and simulation for policy decision support in identity management," in *IEEE International Symposium on Policies for Distributed Systems and Networks*, London, UK, 2009, pp. 17–24.

[3] T. Flanagan, C. Thornton, and J. Denzinger, "Testing harbour patrol and interception policies using particle-swarm-based learning of behavior," in *CISDA-09*, Ottawa, 2009, pp. 1–8.

[4] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *IEEE ICNN 1995*, Piscataway, NJ, 1995, pp. 1942–1948.

[5] M. Reyes-Sierra and C. C. Coello, "Multi-objective particle swarm optimizers: A survey of the state-of-the-art," *Int. Jour. Comp. Int. Res.*, vol. 2, no. 3, pp. 287–308, 2006.

[6] P. Parent and R. Church, "Evolution of geographical information systems as decision making tools," in *GIS '87*, Falls Church, VA, 1987, pp. 63–71.

[7] Natural Rescources Canada, "Geogratis," http://geogratis.cgdi.gc.ca/-geogratis/en/index.html, as seen on March 8, 2010,, 2005.

[8] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[9] C. Thornton, T. Flanagan, and J. Denzinger, "Creating and evaluating goal ordering structures for testing harbour patrol and interception policies," Department of Computer Science, The University of Calgary, Tech. Rep. 2010-955-04, 2010.

[10] M. Harman, "The current state and future of search based software engineering," in *29th ICSE: FoSE*, Minneapolis, MN, 2007, pp. 342–357.

[11] "Sebase: Software engineering by automated search repository," http://www.sebase.org/sbse/publications/, as seen on March 8, 2010, 2010.

[12] J. Denzinger and J. Kidney, "Evaluating different genetic operators in the testing for unwanted emergent behavior using evolutionary learning of behavior," in *IAT 2006*, Hong Kong, 2006, pp. 23–29.

[13] M. Atalla and J. Denzinger, "Improving testing of multi-unit computer players for unwanted behavior using coordination macros," in *CIG'09: Proceedings of the 5th International Conference on Computational Intelligence and Games*, Milano, Italy, 2009, pp. 355–362.

[14] E. de Jong, R. Watson, and J. Pollack, "Reducing bloat and promoting diversity using multi-objective methods," in *GECCO-01*, San Francisco, CA, 2001, pp. 11–18.

[15] J. Denzinger and A. Schur, "On customizing evolutionary learning of agent behavior," in *17th AI*, London, ON, 2004, pp. 146–160.

[16] F. Klügl, R. Hatko, and M. Butz, "Agent learning instead of behavior implementation for simulations - a case study using classifier systems," in *MATES'08*, Kaiserslautern, Germany, 2008, pp. 111–122.