Creating and Evaluating Goal Ordering Structures for Testing Harbour Patrol and Interception Policies

Chris Thornton, Tom Flanagan, Jörg Denzinger
Department of Computer Science, University of Calgary
Calgary, Canada

Internal Report 2010 955-04
Department of Computer Science, University of Calgary

March 23, 2010

Abstract

In this article, we discuss a method for testing policies that guide groups of agents in simulations for interactions with other agents and the environment that reveal weaknesses of these policies. Our method is based on learning interaction sequences using particle swarm systems and has as one crucial component so-called goal ordering structures that are used to guide the learning towards weakness-revealing interactions. Our discussion centers around the different ways a new measuring idea can be integrated into such an ordering structure using the example of testing patrol and interception policies for harbours. Our experimental evaluation reveals that the position of placement of a new measure in an existing ordering structure can greatly influence the testing results, positively and negatively, but mostly mirrors the intuition associated with the placement.

1 Introduction

One of the key uses of simulations is to provide decision support to human decision makers by giving them an idea of the consequences of particular decisions without having to really implement the decisions in the real world (see, for example, [18] or [2]). While often the decision makers evaluate their decisions based on some quality measures, in early stages of their simulations the focus is often on just testing certain decision strategies with regards to producing some expected results. Especially for problems that involve groups of agents and their interactions, simulation runs are first used to see if the individual decision making of agents achieves the intended group behavior (emergent behavior) and if the decisions regarding the environment in which the agents interact produce the intended effects on the agents.

So, while the use of simulations to test if decisions, decision policies, or cooperation concepts achieve the intentions behind them is a very established use of simulations, in the last few years we have seen first approaches that use simulations to find weaknesses in the policies that guide decisions or the cooperation concepts for some or all the agents in a system. In contrast to the established use of simulations for testing, this new use, if automated, requires *searching* for agent behaviors or events and event sequences in the environment that reveal weaknesses of a certain type, which obviously is much harder than just running a single simulation. In fact, in works like [14] or [7], simulation runs are the central piece of the evaluation of candidates for behaviors or events that might reveal weaknesses. But the creation of these candidates and the process of trying to improve them to reach one that really reveals a weakness is the task of a system build around the simulation system. Different evolutionary search (or learning) approaches are the basic methods used in such testing systems and most of these approaches need fitness measures that try to catch the particular test goals while also reflecting the particular search method used. And the underlying simulations usually offer many good candidates for such measures, so that creating a good fitness function is a key problem for these kinds of testing systems.

Testing is not the only area where creating an appropriate fitness function is necessary (see, for example, [6] or [4]), and the experiences from these other areas show that having to bring together several ideas what a good individual (in our case of testing individuals represent behaviors or event sequences) is is usually very difficult and, from a knowledge representation point of view, very indirect, usually introducing additional parameters that need to be chosen well to achieve the wanted results. For many complex testing purposes, we need to be able to define things like subgoals and alternatives within the guidance for a search, which needs to be kept explicit, thus clashing with the idea of a single fitness function needed in many evolutionary approaches. Fortunately, there are search methods that do not need a fitness measure heavily involved in their search controls, needing only to be able to compare two individuals and, in their multi-objective variants, accepting uncomparability. One of these search methods is particle swarm optimization (PSO). In [7], we used PSO to do a simulation based testing of harbour security policies and their implementation.

A key component of the approach of [7] are so-called ordering structures that are used to compare individuals ("particles"). Ordering structures allow to combine several measures by creating hierarchy levels of measures where the measures within one level of the hierarchy are treated like multiple objectives (as in multi-objective optimization, see [20]). And only if two individuals are identical with regard to one hierarchy level, the measures of the lower levels are used in the comparison. Thus, a sequence of subgoals can be translated into the different hierarchy levels and measures for individual agents can be put within a level as different objectives.

In this paper, we present a case study enhancing the ordering structure from [7] by a new idea for a measure. We look at the consequences of placing this new measure into different hierarchy levels and using it as measure for individual agents versus a measure for the whole group of agents. Our experimental evaluation shows that the extensions mostly create the expected effects, including improvements of success rate and speed in finding weaknesses, resp. deteriorating success for extensions that do not make a lot of sense, but we also saw some effects that were surprising. The later were more concerned with order structures that we did not expect much from and the surprises were the fact that we had more testing success than we expected, showing that the underlying testing idea is rather strong.

2 Basic concepts

In this section, we first present the basic concept of testing policies for agents using learning of cooperative behavior in simulations (which we will instantiate in the next section to testing harbour patrol and interception policies). Then we introduce particle swarm system (PSS) based search, especially PSS for multi-objective optimization, which will be the search method we use for testing policies.

2.1 Testing by learning behavior

As already stated, simulations allow to evaluate decision or cooperation policies by having agents that follow these policies interact within a (naturally also simulated) environment. Usually, policies and environment allow for some leeway (or, for agents, individuality), so that there can be agents not following a policy (or only sometimes following it) or several different interpretations of a policy or events in the environment that might occur or not. Also, there are often many possible start situations for a simulation. For an actual simulation run, each agent needs a clear strategy that it follows during the simulation, the events that should occur need to be given to the simulation system and, naturally, a start situation for the run needs to be given. If such a run then reveals an unexpected result, a weakness in the policy employed by the agents (or the set of agents for which a policy is evaluated) is found.

But, due to the leeway that we usually have, it is rather unlikely that the first simulation run will result in finding a weakness. Human decision makers will therefore vary all of the parameters that in the simulation system express the possible leeway to produce several simulation runs and if none of those runs reveal weaknesses at one point the decision makers will be satisfied that the policy they test works. This naturally does not mean that there is no weakness in the policy. In fact, it is highly dependent on the decision makers and the available time how likely it is that a certain weakness of a policy is detected or not. Therefore, an automated testing approach is favorable, since it at least does not depend on whether a novice or expert decision maker is doing the testing and whether the human decision maker has a good or bad day.

Figure 1 describes the general structure of an automated policy testing system on top of a simulation system. At the core of such a system is the environment $\mathcal{E}nv$ simulated by the simulation system. Within $\mathcal{E}nv$, there can be three kinds of agents. The set $A_{pol} = \{\mathcal{A}g_{pol,1},...,\mathcal{A}g_{pol,m}\}$ is the set of agents that represent the policy that we are testing. "Represent" in this context means that these agents follow the policy in all their actions. The agents in the set $A_{other} = \{\mathcal{A}g_{other,1},...,\mathcal{A}g_{other,k}\}$, which can be empty, are agents that do not follow the policy, are not under the control of the tester, but still participate in the simulation. Finally, the set $A_{test} = \{\mathcal{A}g_{test,1},...,\mathcal{A}g_{test,n}\}$ represents the agents in the simulation that the tester can control. Again, A_{test} can be empty (if the tester is only allowed to create events in the environment, see below), but -as already stated- many policies allow for some leeway and the agents in A_{test} model these agents for which

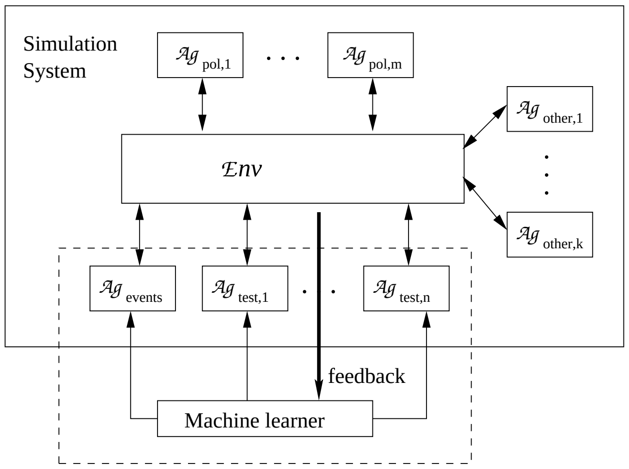
Simulation System
$\mathcal{A}g_{\text{pol},1}$
$\mathcal{A}g_{\text{pol},m}$
$\mathcal{A}g_{\text{other},1}$
$\mathcal{E}nv$
$\mathcal{A}g_{\text{other},k}$
$\mathcal{A}g_{\text{events}}$
$\mathcal{A}g_{\text{test},1}$
$\mathcal{A}g_{\text{test},n}$
feedback
Machine learner

Figure 1: Testing policies by learning cooperative behavior

this leeway is to be explored. During a simulation, not only the actions of the agents create events. It is also possible that events in the environment are just happening, for example, it can become night (or, after that, day again) in the simulation. And some of these events fall into the leeway area that we want to explore, like, for example, it snowing in a traffic simulation. For such events that can be caused by the tester, we model their scheduling by having an explicit agent, $\mathcal{A}g_{events}$, that is able to tell the simulation system when particular events are happening.

With a system as sketched above, testing the policies represented by the agents in A_{pol} for weaknesses can be automated by having a machine learner that learns behaviors for the agents in A_{test} and for $\mathcal{A}g_{events}$ that reveal a weakness when these agents interact with $\mathcal{E}nv$ and the other agents in the simulation system. A learning approach using feedback from simulation runs is necessary, since usually there are a large number of behaviors for the agents in A_{test} and also for event sequences for $\mathcal{A}g_{events}$, so that it is not possible to systematically try out all possibilities. Instead, similar to human decision makers, a learner needs to evaluate the results of simulation runs, adjust the behaviors of the agents in A_{test} and of $\mathcal{A}g_{events}$, and repeat this cycle until a weakness is found or the resources set aside for testing are used up.

In the following, we will denote with $Act_{test,i}$ the set of possible actions that an agent $\mathcal{A}g_{test,i}$ can perform and with Act_{events} the events $\mathcal{A}g_{events}$ can invoke in the simulation system. During a simulation run, the learner receives feedback in the form of environment states e, so that, from the learner's perspective, a simulation run is represented by a sequence (or trace) e_0,e_1,...,e_x of such states. If the sequence of actions taken by $\mathcal{A}g_{test,i}$ is (a_{i1},t_{i1}),...,(a_{il_i},t_{il_i}) with $a_{ij} \in Act_{test,i}$ and t_{ij} the time (within the simulation run) that $\mathcal{A}g_{test,i}$ starts performing a_{ij}, then obviously the environmental state after each action is performed by any of the $\mathcal{A}g_{test,i}$s should be included into e_0,e_1,...,e_x (and the same should be true for the states after each event triggered by $\mathcal{A}g_{events}$). But there can be more environment states that are considered by the learner.

There are many possibilities how the machine learner can be realized. Naturally, the concrete agent architecture(s) used for the $\mathcal{A}g_{test,i}$s has some influence on how the learning has to be performed, as have the application area and the simulation system. The machine learning method we used in our experiments is based on the concept of particle swarm systems, that we introduce in the next subsection.

2.2 Particle Swarm Systems

Particle Swarm Systems (PSS, see [11]) are inspired by physics and biology, enhancing the idea of a moving particle with the attraction behavior of members of a swarm, to perform a search in a solution space (or an extention of such a space). In a PSS, the search state is represented by a set of l *particles* p_i each of which is characterised by its current position pos_i, its current velocity v_i, and its best position $best_i$ in the past. The position of a particle is usually a vector of continuous variables that represent a solution to the instance of the search problem the PSS is aimed at solving and discrete variables are dealt with by rounding a continuous variable to the nearest allowed value. In the basic case of a PSS there is a single function f describing the quality of a solution/position and the goal of the search is to find a solution that is as good as possible with regard 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 so-called *cognitive learning factor*, C_2 the so-called *social learning factor* and r_1,$r_2 \in [0, 1]$ are random values chosen by the search control. *Best* is the best position the whole swarm has found so far. If a particle reaches a new best position, i.e. $f(pos_i^{new})$ is better than $f(best_i)$, then $best_i$ is updated, i.e. $best_i^{new} = pos_i^{new}$, else it stays, i.e. $best_i^{new} = best_i$. From the point of view of a particle, a sequence of updates has it *flying* through the solution space and the whole algorithm terminates either after a given number of update rounds (with $Best$ being the output of the system) or if $Best$ fulfills certain conditions.

For many applications, including in a certain sense the testing of policies we look at in this article, there is no single function f describing what is searched for, but instead we have a vector $\vec{f} = (f_1,...,f_q)$ of quality or goal functions, which moves these applications into the area of multi-objective optimisation. In most of these cases, we are then not interested anymore in a single solution (although in our application this is not exactly the case, see Section 3.2), since there is not one position that is optimal for all goal functions, instead positions that are very good for one f_i often are not so good for an f_j (with $i \neq j$). A key concept of multi-objective optimisation (and for our ordering structures) is the *domination* of one solution x_1 over a solution x_2, denoted by $x_1 \succ_{\vec{f}} x_2$ which is defined by $f_i(x_1) \geq f_i(x_2)$ for all i (if our goal is to maximise all functions in $\vec{f}$). The subset PF of all possible solutions Sol to a multi-objective optimisation 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.

PSS is among the search concepts that can be easily extended to deal with multi-objective optimization, in fact there are quite a few variants for this extention (see [20] for an overview). Fortunately, for our application we can use a rather primitive variant. We extend the definition of a particle to a triple $p_i = (pos_i,v_i,Ownbest_i)$, where the set $Ownbest_i$ is used to record 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 select the position $Best$ in Equation (1) out of 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 (1). After the new position of p_i is created, it is checked if it is dominated 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 is finished if a given time limit or number of updates is reached or the union of all $Ownbest_i$s fulfills certain conditions.

3 Testing of harbour security policies

In this section, we will first present our application problem, testing harbour patrol and interception policies, and the simulation system we use to do so. Then we present the instantiation of our testing approach from Section 2.1 using particle swarm systems to implement the machine learner, after which we will focus on different ordering structures.

3.1 Harbour simulations for patrol and interception policies

Due to the large number of goods that pass through them, large commercial harbours represent an important part of a country's infrastructure that needs to be protected from harm. Some harbours additionally house military installations that add to the possible targets for terrorists. And the safety of harbours needs to be achieved with a lot of legitimate users of these harbours following their own agendas and requiring access to various parts of a harbour. As a consequence, various policies for different types of harbour users and defenders are needed to ensure harbour security and the interplay of these policies is in no way trivial and finding weaknesses is an important task for the policy makers. Even more, already creating the individual policies can be very difficult, due to various outside limitations on them, like resource limitations in the number of defenders, and a wide variety of events, as, for example, weather events.

Harbours are also a very good example for why we need to use simulations to test policies. It is definitely too expensive to evaluate any policy in a real harbour, so that the use of simulations is a must. In this article, as in [7], we are interested in testing one particular policy around harbours, namely the policy guiding the patrol and interception vessels whose tasks are to detect threats to the harbour and to intervene to neutralize these threats. Using our notations from Figure 1 and Section 2.1, the patrol and interception vessesls are the $\mathcal{A}g_{pol,i}$ agents and the important features of these agents that need to be simulated are their sensor capabilities and the movement capabilities, the later essentially boils down to the speed with which vessels do move.

The goal of any policy for the agents in A_{pol} is to detect any possible threat to installations (or ships) in the harbour, investigate a possible threat and, if the threat is real, neutralize the threat. The investigation part is necessary, since there are usually many possibilities how a legitimate harbour user can look like a threat and "neutralizing" a harmless pleasure boat that drifted a little bit out of the way, for example, is not an acceptable practice.

In our notations, this means that we have as agents in A_{other} the legitimate users of a harbour and our $\mathcal{A}g_{test,i}$ are the agents that try to attack the policy used by the defenders in A_{pol}. Naturally, our $\mathcal{A}g_{pol,i}$ can initially not distinguish agents from A_{other} and A_{test}. In our experiments, the agents in A_{test} will try to get one of them into a particular spot (target) in the harbour, which could be the docking slip of a particular ship or a position from which a certain harbour facility could be destroyed.

The environment $\mathcal{E}nv$ at the center of our simulation system uses GIS (Geographic Information System) technology (see [17]) enhanced to simulate movement and (sensor-based) perception of all agents. The necessary geographical data for the GIS for our experiments came from the National Topographic Data Base (of the Government of Canada) which is available from [16]. 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 1/10ths of a second 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.

3.2 Using PSS to test harbour patrol and interception policies

For instantiating our general policy testing scheme from Section 2.1 to test harbour patrol and interception policies, we need an agent architecture to realize the agents in A_{test} and we need a machine learner that can learn behaviors for agents using this agent architecture. If we look at the tasks the attack agents in A_{test} have to perform to test harbour security as described in the last subsection, then an $\mathcal{A}g_{test,i}$ essentially moves around in the harbour. When an $\mathcal{A}g_{test,i}$ reaches the target spot, it obviously will do something that our tested policy is supposed to prevent, but just reaching the spot reveals that there is a weakness in the policy, so that we do not really have to care what happens after that. If an $\mathcal{A}g_{test,i}$ is intercepted and neutralized, there might also be some things this $\mathcal{A}g_{test,i}$ can do (like explode in order to take the $\mathcal{A}g_{pol,j}$ that does the intercept out, or resist arrest), but for testing the policy guiding the agents in A_{pol} what really is of interest is if the $\mathcal{A}g_{pol,j}$ can continue doing tasks and if yes, how long it takes for it to continue after an intercept. So, also for intercepted $\mathcal{A}g_{test,i}$ we do not need any additional actions than moving around in the harbour.

So, the agent architecture of an $\mathcal{A}g_{test,i}$ needs to produce a series of movements and, since timing is clearly something that is needed to avoid detection, the speed with which these movements are performed. As stated in [7], initial experiments using movement actions (as in [5]) showed that having the learner figure out how to avoid obstacles is not very successful, since the learn focus is not on finding weaknesses of the security policy anymore. Therefore we moved to an agent architecture that has the agent move between waypoints (with a certain speed; this means that $Act_{test,i}$ is the set of all waypoint-speed pairs). Since avoiding obstacles is still something that should not be figured out by the learner, we have as part of the agent architecture that an agent uses a path planner to determine how to get from one given waypoint to the next one. This path planner (we used a standard one, as described in [9]) creates additional, low-level waypoints that minimize the path between two high-level waypoints and an agent, when given a sequence of high-level waypoints, first creates the additional low-level waypoints and then moves from waypoint to waypoint (following the sequence of combined high- and low-level waypoints).

Given the agent architecture from above, the behavior of an agent in A_{test}, from the point of view of the learner, can then be described as a sequence of high-level waypoints (x,y) together with the speed *speed* of the agent between each pair of following waypoints. In our simulation, *speed* is a number between 0.1 and 1 indicating the throttle position of the vessel while x and y are real numbers. In our current version of our test system, we are not creating any events in the environment, so that we do not need an architecture for an agent $\mathcal{A}g_{events}$. So, a behavior is a sequence of numbers, which now allows us to look at particle swarm systems to learn such a sequence.

In order to use PSS for our problem, a particle position (or attack strategy) in our system has the general form

$(((x_{1,1},y_{1,1},speed_{1,1}),...,(x_{1,l_1},y_{1,l_1},speed_{1,l_1})),$

$...,$

$((x_{n,1},y_{n,1},speed_{n,1}),...,(x_{n,l_n},y_{n,l_n},speed_{n,l_n}))),$

which is, as mentioned above, a sequence of high-level waypoints with speeds for each of the agents in A_{test}.

As stated before, the learner evaluates a particle by having each agent in A_{test} take its sequence of waypoints (i.e. $((x_{i,1},y_{i,1},speed_{i,1}),...,(x_{i,l_i},y_{i,l_i},speed_{i,l_i}))$ for $\mathcal{A}g_{test,i}$) and apply this sequence in a simulation run of the harbour simulation with the agents from A_{pol} implementing the patrol and interception policy that is tested and the agents from A_{other} (in our current version, A_{other} is the empty set; this makes it on the one hand side more difficult for the agents in A_{test}, because they can not use any agents from A_{other} to hide behind, but it is also easier on us, since we did not have to come up with behaviors for the agents in A_{other}) as described above. 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 center of our goal ordering structures that this article is about, we will look more closely at this part of our PSS in the next subsection.

Our particles are updated as described in Section 2.2. Naturally, it can happen that the position update results in a waypoint that puts an agent on land or in an invalid speed. If a waypoint is not over water, in the simulation run we substitute this waypoint by the nearest point to it that is over water. If we have a speed value that is not between 0.1 and 1, we round to the nearest point in the interval which favors high-speed manouvers. 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 at most 600 meters 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 if either the attack objective is fulfilled (i.e. an attacker $\mathcal{A}g_{test,i}$ reached the target spot in the harbour), or all attackers have been intercepted (i.e. the policy was successful), or all attackers are at the end of their sequence of waypoints.

In [7] we also proposed an additional possibility to create a new position for a particle that is based on the idea of targeted operators in genetic algorithms from [5]. If the attack strategy represented by a particle position leads to a point in the simulation where all but one attack agent have been taken out of the simulation (i.e. they have been intercepted by the agents in A_{pol}), then we can update the particle so that the next waypoint for the agent after the waypoint when all other agents are intercepted is changed to the target spot that the policy we test wants to protect. This reflects the hope that now the way is clear and that this hope should be tried out.

3.3 Goal ordering structures for breaking harbour security policies

As stated in the last subsection, the one component of a PSS that we have not instantiated, yet, is the goal function for the optimization that a PSS performs. And, as stated in the introduction, for our application there is no obvious candidate, in fact there are many possible things to measure in a simulation run that might be useful in stearing the learning process towards finding a weakness in the policy implemented by the agents in A_{pol}. Naturally, a particle position representing a behavior strategy of the attackers in A_{test} that results in having one $\mathcal{A}g_{test,i}$ reaching the target spot (without being intercepted) reveals a weakness in the policy and therefore fulfills the goal of or learning process, but we cannot expect the learner to have a particle with such a position among the initial positions for the particles.

So, in order to compare particle positions, we need measures out of the environmental states of a simulation run that tell us how near the behaviors of the $\mathcal{A}g_{test,i}$s come to achieving the ultimate goal of finding a weakness, or, in the case of PSS, we need to be able to compare two positions with regard to which one is more on the way to achieving the ultimate goal. Initial ideas for measures are the number of agents in A_{test} not being intercepted, nearness of $\mathcal{A}g_{test,i}$s to the target, or the distance of the $\mathcal{A}g_{test,i}$s to the defenders in A_{pol} (with the later being the better the larger the distance). But all of these measures capture only aspects of the ultimate goal and there are also additional problems due to the fact that we have several agents in A_{test}, namely how to combine the measures from the different agents (averaging them, using the best/worst value among them, summing them up, or some other combination idea). And, if the policy has the agents in A_{pol} (or at least some of them) near the target spot, then trying to get near this spot and trying to stay away from defenders are rather contradictory measures.

To deal with these problems in the context of PSS, in [7] we created the concept of so-called goal ordering structures that create a hierarchy of goal measures, where each hierarchy level represents a set of measures that are treated like the objectives of a multi-objective optimization and the different levels then essentially represent a lexicographic combination of orderings. This is a generalization of an idea from [15]. While [7] introduced goal ordering structures, it only presented one such structure and did not explore the possibilities around them at all. This exploration is what we will be doing in this article.

The general idea of a goal ordering structure $\rhd$ can be formally described as follows. For two particle position vectors pos_1 and pos_2, that need to be compared, a goal ordering structure has the form

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

(or $(\vec{f_1},...,\vec{f_u})$ for short), where f_{ij} is a quality function assigning an integer to a trace $e_0,e_1,...,e_x$ of environmental states produced by the strategy for the attack agents represented by a position when applied in $\mathcal{E}nv$ interacting with the other agents. If $\rhd$ denotes the ordering that is created by this ordering structure, then we have

$pos_1 \rhd pos_2$,
if
$pos_1 \succ_{\vec{f_1}} pos_2$, or
$pos_1 =_{\vec{f_1}} pos_2$ and $pos_1 \succ_{\vec{f_2}} pos_2$ or
... or
$pos_1 =_{\vec{f_1},...,\vec{f_{u-1}}} pos_2$ and $pos_1 \succ_{vecf_u} pos_2$.
$pos_1 =_{\vec{f_i}} pos_2$ in this context means that pos_1 and pos_2 have an identical quality value in each of the measures f_{ij} in $\vec{f_i}$ (and $=_{\vec{f_1},...,\vec{f_i}}$ is short for $=_{\vec{f_1}}$ and $=_{\vec{f_2}}$ and ... and $=_{\vec{f_i}}$). As already stated, this essentially represents a lexicographical combination of multi-objective domination orderings, which -due to the partiality of the domination orderings- is itself a partial ordering, so that two positions might not be comparable. Due to this incomparability, we have to use a multi-objective version of PSS although we have a single ultimate goal for the PSS to search for.

In Section 4, we instantiate the goal ordering structure concept for our application of testing harbour security policies using 5 measures, resp. measure groups: If e_0,...,e_x is the trace produced by the simulator run for a particle position pos, then

$$f_{intercept}((e_0, ..., e_x), pos) = \begin{cases} 0, & \text{if there is an } j \text{, such} \\ & \text{that all } \mathcal{A}g_{attack,i} \\ & \text{are intercepted in } e_j \\ 1, & \text{else} \end{cases}$$

with $pos_1 \succ_{intercept} pos_2$, if
$f_{intercept}((e_0, ..., e_x), pos_1) > f_{intercept}((e_0, ..., e_x), pos_2)$.

$$f_{success}((e_0, ..., e_x), pos) = \begin{cases} 1, & \text{if there are } j, i \text{, such} \\ & \text{that } \mathcal{A}g_{attack,i} \text{ reached} \\ & \text{the target spot in } e_j \\ 0, & \text{else} \end{cases}$$

with $pos_1 \succ_{success} pos_2$, if
$f_{success}((e_0, ..., e_x), pos_1) > f_{success}((e_0, ..., e_x), pos_2)$.

$$\begin{aligned} f_{dist,i}((e_0, ..., e_x), pos) \quad &= \quad \sum_{j=1}^{\lfloor x/100 \rfloor} dist(e_{100j}, \mathcal{A}g_{attack,i}) \\ &\quad +dist(e_x, \mathcal{A}g_{attack,i}) \end{aligned}$$

where $dist(e, \mathcal{A}g_{attack,i})$ is the length of the shortest path created from the position of $\mathcal{A}g_{attack,i}$ in e to the target spot (again computed using path

finding). We define $pos_1 \succ_{dist,i} pos_2$, if
$f_{dist,i}((e_0, ..., e_x), pos_1) < f_{dist,i}((e_0, ..., e_x), pos_2)$.

$$\begin{aligned} f_{hide,i}((e_0, ..., e_x), pos) &= \sum_{j=1}^{\lfloor x/100 \rfloor} ndist(e_{100j}, \mathcal{A}g_{attack,i}) \\ &\quad +ndist(e_x, \mathcal{A}g_{attack,i}) \end{aligned}$$

where $ndist(e, \mathcal{A}g_{attack,i})$ is the shortest distance between $\mathcal{A}g_{attack,i})$ and any of the vessels in A_{tested} in e. We define $pos_1 \succ_{hide,i} pos_2$, if
$f_{hide,i}((e_0, ..., e_x), pos_1) > f_{hide,i}((e_0, ..., e_x), pos_2)$.
Finally, in our experiments we will also use a variant of the last measure to compare using one measure for each agent in a level with a combined measure for all agents, namely

$$f_{hidesum}((e_0, ..., e_x), pos) = \sum_{i=1}^{n} f_{hide,i}((e_0, ..., e_x), pos)$$

where $pos_1 \succ_{hidesum} pos_2$, if
$f_{hidesum}((e_0, ..., e_x), pos_1) > f_{hidesum}((e_0, ..., e_x), pos_2)$.

We used the first three measures in [7] to define the ordering structure $\rhd_{base}$ as
$(\{f_{intercept}\}, \{f_{dist,1}, ..., f_{dist,n}\}, \{f_{success}\})$.
This ordering structure has an attack strategy in which all attackers are intercepted as always worse than a strategy with some of the attackers "alive" at the end of the simulation run, due to using $f_{intercept}$ as the sole component of the first level of the sequence (essentially as a first subgoal for the learner to achieve). Since, as mentioned above, it is easily possible to achieve the survival of attackers by simply staying away from the harbour (or at least the patrol and interception vessels), the second element of the sequence uses the $f_{dist,i}$s to drive the search process towards attack strategies that get near to the target spot, allowing for individual attackers making progress individually (with respect to the global search of the learner) because of treating this component as a multi-objective component using the results of each attacker as a single objective. The third element in the sequence of $\rhd_{base}$ favours successful attacks over unsuccessful ones.

With the introduction of new measures, more precisely the $f_{hide,i}$s and $f_{hidesum}$, the question now becomes where to put them into an extension of the ordering structure $\rhd_{base}$ to help with the search. Here, helping can have two meanings, namely on the one hand side to speed up learning successful attacks (or, given that we will have to use resource limits, having a higher

percentage of learning runs that are successful with regard to the ultimate goal), but, given that $\succ_{hide,i}$ and $\succ_{hidesum}$ aim at keeping attackers away from the tested agents, it can also mean directing the search towards new kinds of attacks[1]. An additional question is to see which of the two rather similar measures is better suited, individual measures for agents or a combination (here an accumulation) of these individual measures. If we look at $\rhd_{base}$, then placing the new measure(s) before or with the first component or after or with the last component does not make a lot of sense, so that we essentially are left with before the second component, in the second component or after the second component. Putting them into the second component produces the problem mentioned before, namely having measures that essentially contradict each other. Therefore we will be looking in the next section at the following four new goal ordering structures: The definition of $\rhd_{hidebefore}$ is
$(\{f_{intercept}\}, \{f_{hide,1}, ..., f_{hide,n}\}, \{f_{dist,1}, ..., f_{dist,n}\}, \{f_{success}\})$,
$\rhd_{hideafter}$ consequently is defined as
$(\{f_{intercept}\}, \{f_{dist,1}, ..., f_{dist,n}\}, \{f_{hide,1}, ..., f_{hide,n}\}, \{f_{success}\})$,
and similarly we have $\rhd_{hsumbefore}$ defined as
$(\{f_{intercept}\}, \{f_{hidesum}\}, \{f_{dist,1}, ..., f_{dist,n}\}, \{f_{success}\})$,
and, finally, $\rhd_{hsumafter}$ as
$(\{f_{intercept}\}, \{f_{dist,1}, ..., f_{dist,n}\}, \{f_{hidesum}\}, \{f_{success}\})$.
Given the particular aspects of a simulation run that the different measures look into, it is our expectation that the ordering structures where we put the new measures before the distance to the target measures will not be very successful, since they will allow the learner to keep generating attack strategies where the agents in A_{test} stay away from the harbour. On the other side, we hope that the ordering structures where we put the new measures behing the distance component will show an improvement compared to $\rhd_{base}$. And goal ordering structures allow to explicitly represent such plans for guiding the PSS, which makes them very interesting for testing policies in simulations.

4 Experimental evaluation

In order to present our experimental evaluation of the different goal odering structures from the last section, we will first describe the different policies (and harbours) that we used in our evaluation and the general set-up of the simulation system and the testing system. Then we will first present and

[1]Like attacks that do not sacrifice attackers, see Section 4.

comment on the quantitative data from our experiments to see whether our expectations for the different ordering structures are fulfilled. Finally, we will take a closer look at some of the attack strategies our testing system found with the different ordering structures.

4.1 Experimental set-up

In our experiments, we used two different harbour patrol and interception policies. Since we were naturally not able to get any real policies used for the two harbours we have in our simulations, due to understandable security concerns[2], we had to make up the two policies and we tried to make them both reasonable and not perfect (with the not perfect part not exactly being difficult). Both policies have to be considered hugh-level patrol and interception policies suitable for any harbour and any target spot. Therefore we have to provide also some information on how these general policies are instantiated for the individual harbours.

The first patrol and interception policy divides the agents in A_{pol} into two subtypes, namely patrollers and interceptors, with using the patrollers to notice potential attackers/intruders and alert the interceptors that then approach a potential intruder, identify it and, if necessary, take it out. A patroller updates an alarmed interceptor about the course of the potential intruder as long as it is in its sensor range. If a potential intruder comes close enough to a patroller to be identified, then the patroller will take it out. With the exception of this case, patrollers stay to their predetermined route through the harbour. The interceptors wait at predefined positions in the harbour and only become active when alarmed by a patroller. When active, an interceptor determines the best position to come near to a potential intruder, based on the information from the patroller. If the intruder is not detected at that position then the interceptor returns to its waiting position. By splitting the agents in A_{pol} into two types, resources can be used in a more dedicated fashion, with the patrollers more sspecialized in detection and the interceptors with man power and other resources to deal with intruders. This is policy $pat - int$.

Our second policy, $all - pat$, does not distinguish the agents in A_{pol}. All patrollers follow a circuit around the harbour (like a patroller in $pat - int$) and the available vessels are evenly spaced on this circuit. If an agent detects a potential intruder, the available agent closest to the intruder is sent to

[2]We also were not able to get any old, not used anymore, policies, due to security concerns, again.

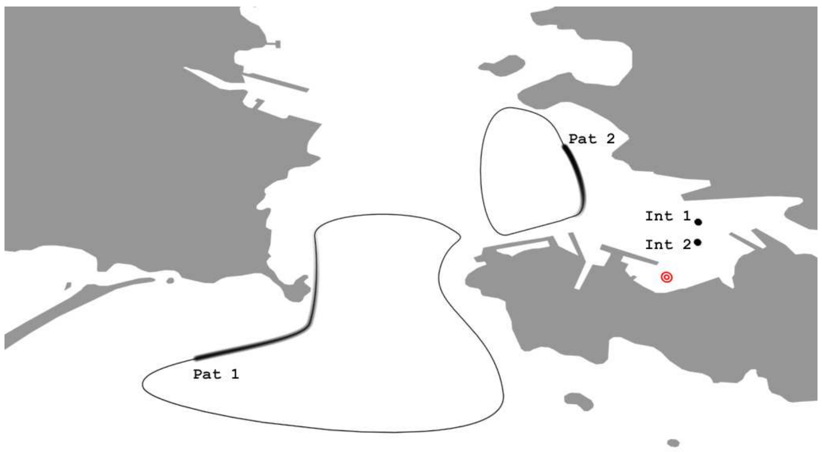

Figure 2: The patrol policy $pat - int$ for Esquimalt harbor

identify the boat and if this identification has as result the need to intercept then this agent will intercept the intruder.

In our simulation system, we have two possible environments, namely 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 inactive positions near the dock adjacent to the target spot. The target is placed deep inside the harbour behind a pier (see Figure 2 for the position of the target, which is the same for all our experiments with this harbour, and the patrol pattern). 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 (Patroller 2 in Figure 10) doing its patrol route relatively near to the target. Figure 10 shows also the positions for the two interceptors when not active, near the

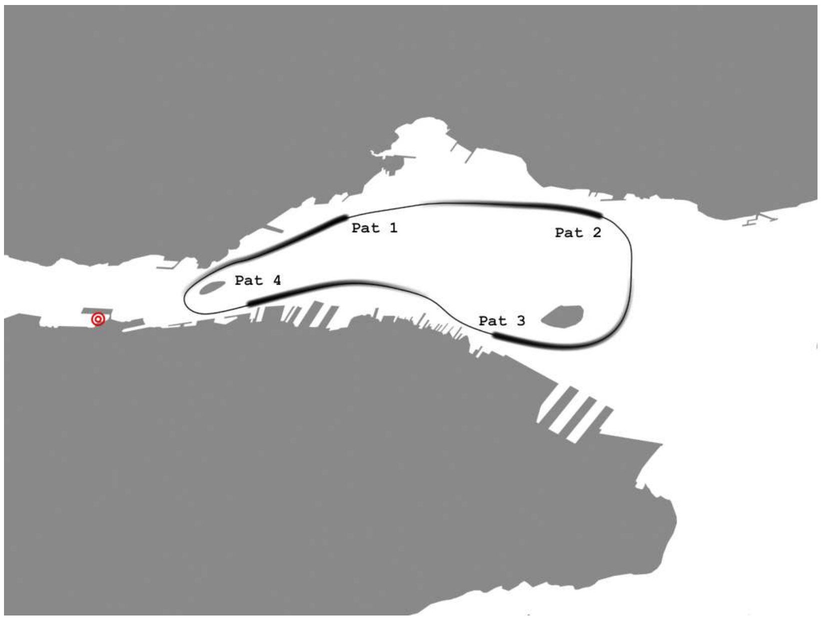

Figure 3: The patrol policy $all - pat$ for Halifax harbor

target, which is indicated on the right of the picture (again, this target spot is the same for all our experiments for this harbour).

For policy $all - pat$ and Esquimalt we used 4 patrollers circling around inside the harbour. We also used 4 patrollers for the Halifax scenarios . As can be seen in Figure 3, the patrollers have to do a rather large circuit on their patrols making them vulnerable for good timed attacks. In all scenarios, the sensor perception by an $\mathcal{A}g_{pol,i}$ was implemented as a circle around the agent with a radius of 300 meters. In order to identify whether an agent is a threat or not, an $\mathcal{A}g_{pol,i}$ needs to get within 20 meters of this agent.

As stated before, the two policies and their instantiations for the two harbours have weaknesses, even if we limit the number of agents in A_{test} (to avoid the easy attack of simply having too many attackers and therefore easily overwhelm the defenders). The limited experiments in [7] 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 and then having an attacker cruising by to the target. We consider the sneaking by without use of decoys as the more dangerous weakness (since there are additional measures involving other defense units than the agents in A_{pol} that can be activated when the presence of any intruder is detected) and therefore part of our goal in looking at new ordering structures was to focus our testing system on finding more attack strategies that accomplish an attack without being noticed before the attack goal is reached.

In our experiments, we set the parameters of our testing system as follows: the PSS parameters were $W = 0.8$, $C_1 = 0.2$, and $C_2 = 0.4$ (this is similar to [7]). The number of waypoints for an $\mathcal{A}g_{test,i}$ in an attack strategy was 10 and we used 20 particles. We had a maximum of 100 position updates per particle and every entry in Tables 1 and 2 is based on performing at least 100 runs of the testing system (due to the random factors involved in the PSS, repeating testing runs, without special measures, results in different outcomes, naturally). While [7] showed that the approach has merrit, having only 3 runs per scenario is obviously not a good foundation, which is why we took the opportunity to do a much more careful experimental evaluation in this article. Initial experiments showed that the targeted particle update favored the detection of weaknesses using decoys, so that we are not using this feature in our experiements.

Given the number of testing runs necessary to populate the tables, we needed to speed up the whole system and use a cluster of, unfortunately, heterogeneous machines. We achieved a substantial speed-up by re-implementing the whole system in C++ (instead of the original Python), so that doing an update of the 100 particles, including the simulation runs to evaluate them takes around one minute on an iMac with a 2.4 GHz Intel Core 2 Duo processor running MacOSX 10.5 (which is the configuration most of the machines in our cluster have). Due to the heterogenety of the cluster, we use the average number of particle updates (generation) as speed measure.

4.2 Quantitative analysis

Table 1 reports the success rates of our testing system for the different test scenarios and the different goal ordering structures (using either 2 or 3 agents in A_{test}). 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 (despite not using the targeted update), so that the basic goal ordering structure is already rather powerful in helping to reveal weaknesses in the policies. But the ordering structures that put the new measure in a level after the distance-to-target

Harbour:	Esquimalt				Halifax			
Policy:	$pat-int$		$all-pat$		$pat-int$		$all-pat$	
Agent numbers:	2	3	2	3	2	3	2	3
$\rhd_{base}$	67.3	74.5	46.6	57.1	77.2	84.2	68.1	92.1
$\rhd_{hideafter}$	64.0	75.5	45.7	57.7	75.5	89.6	79.5	85.4
$\rhd_{hsumafter}$	67.3	77.6	45.7	58.2	73.3	88.4	75.5	88.5
$\rhd_{hidebefore}$	23.6	38.6	13.6	28.6	44.9	68.3	66.7	82.6
$\rhd_{hsumbefore}$	9.6	34.2	15.2	19.7	53.4	68.4	68.8	89.2

Table 1: Comparisons between goal ordering structures: success rates in percent

Harbour:	Esquimalt				Halifax			
Policy:	$pat-int$		$all-pat$		$pat-int$		$all-pat$	
Agent numbers:	2	3	2	3	2	3	2	3
$\rhd_{base}$	17.2	16.2	16.6	15.6	23.2	21.0	18.4	16.5
$\rhd_{hideafter}$	15.8	17.5	15.9	15.1	23.7	20.7	18.2	16.4
$\rhd_{hsumafter}$	16.9	16.4	14.9	14.2	22.8	19.6	18.2	16.8
$\rhd_{hidebefore}$	13.2	16.9	10.9	14.5	25.5	22.1	19.1	14.6
$\rhd_{hsumbefore}$	18.1	14.5	15.1	11.5	23.5	22.6	15.8	13.9

Table 2: Comparisons between goal ordering structures: average successful generation

measures were able to improve the success rates in half of the scenarios, with the highligt 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 $\rhd_{base}$, see Table 2. The worst difference in success rate was 3 percent. As expected, having either of the new measures before the distance-to-target measures was not very successful, although we were surprised that there were so many successful runs and that those successful runs usually needed clearly less particle updates than the other ordering structures (see Table 2). We will look into this in more detail in the next subsection.

As Table 2 shows, the average number of particle updates until a successful attack is found, averaging only over successful runs of the testing system, in general favors $\rhd_{hideafter}$ 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, although not really substantial ones.

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, i.e. 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, but 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. But, as Table 1 shows, with regard to success rate, the $f_{hidesum}$-variant is better than the associated $f_{hide,i}$-variant 9 times versus 6 times the other way around (with 1 tie). If we look at Table 2, 10 times $f_{hidesum}$ is faster than $f_{hide,i}$, with 5 scenarios the other way around (again with 1 ties). 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, to allow for this), which explains the unexpected performance of the $f_{hide,i}$s.

4.3 Selected attack strategies

To look into the differences between the five goal ordering structures, Figures 4 to 8 present representations of successful attack strategies for policy $pat - int$ for Esquimalt harbour using two attackers (movies showing the whole behaviors for all the examples in this section can be found at http://www.cpsc.ucalgary.ca/∼denzinge/papers/Movies/harbour/overview.html). While Figures 5 to 8 are typical results for the new ordering structures, Figure 4 represents a found attack strategy that highlights that $\rhd_{base}$ still allows for decoys. Looking at Figure 4, Attacker 1 (Atk 1) is initially identified by Patroller 1 (Pat 1), and according to the policy, the closest interceptor will come out and deal with the threat, as seen in the second frame. Interceptor 2 (Int 2) reaches Attacker 1, disables it, and returns to its home position. In the fourth frame, Attacker 2 (Atk 2) begins to enter the harbour, and heads toward the target zone. It slips by Patroller 2 (Pat 2) without being detected, and since both interceptors are dormant, will not be spotted as it makes its way to the target. However, the waypoints take it beyond the target and into the small inlet before commencing its attack run, as seen in the sixth frame. Continuing the learning might get rid of this "detour", but,

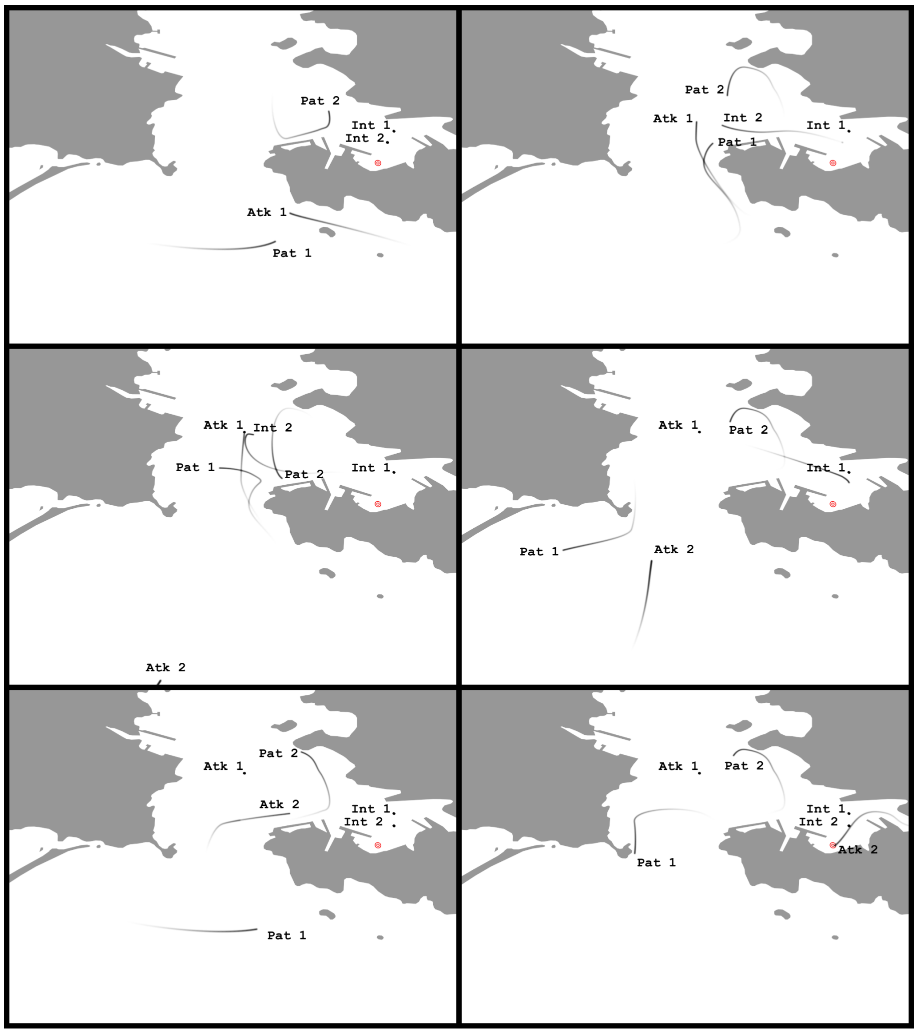

Figure 4: An attack scenario for $pat - int$ in Esquimalt found using $\rhd_{base}$

as stated before, our testing system stops when the ultimate goal is reached.

Figures 5 and 6 show typical attack strategies for the goal ordering structures that put the new measures after the distance-to-target measures. In Figure 5, we see both attackers move closer to the harbour from their starting points, but Attacker 1 does not enter it. Instead, it stays outside of the sensor envelope of Patroller 1, while Attacker 2 makes another timing based attack to pass Patroller 2 without being detected before arriving at

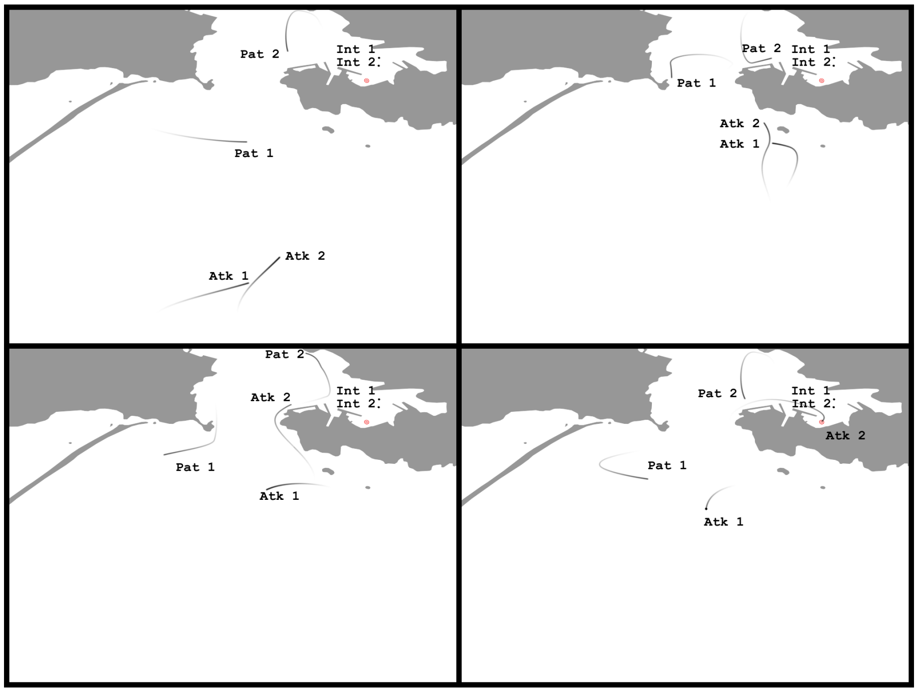

Figure 5: An attack scenario for $pat{-}int$ in Esquimalt found using $\rhd_{hideafter}$

the target. Since this is a timing attack, it is only necessary that one attacker performs the attack, and the goal ordering structures $\rhd_{hideafter}$ and $\rhd_{hsumafter}$ prefer attack strategies that keep "unnecessary" attackers far away. This can also be seen in Figure 6, where Attacker 1 makes a direct attack on the target. It is important to note (see later) that the waypoints for Attacker 1 are exactly those needed to reach the target, and they form a nearly straight line between Attacker 1's starting position and the target. And, as already stated, Attacker 2 remains very close to its starting position outside the harbour.

Figures 7 and 8 represent typical successful attack strategies found using the goal ordering structures that put the new measures before the distance-to-target measures. In Figure 7, Attacker 2 can be seen briefly in the first frame, but afterwards it begins its run out to sea, and proceeds rapidly away from the harbour. Attacker 1 makes a slow zig-zag type of approach to the harbour, then increases the throttle (as can be seen by the stretching of the trail in the third frame) to begin its timing attack on the target. In Figure 8,

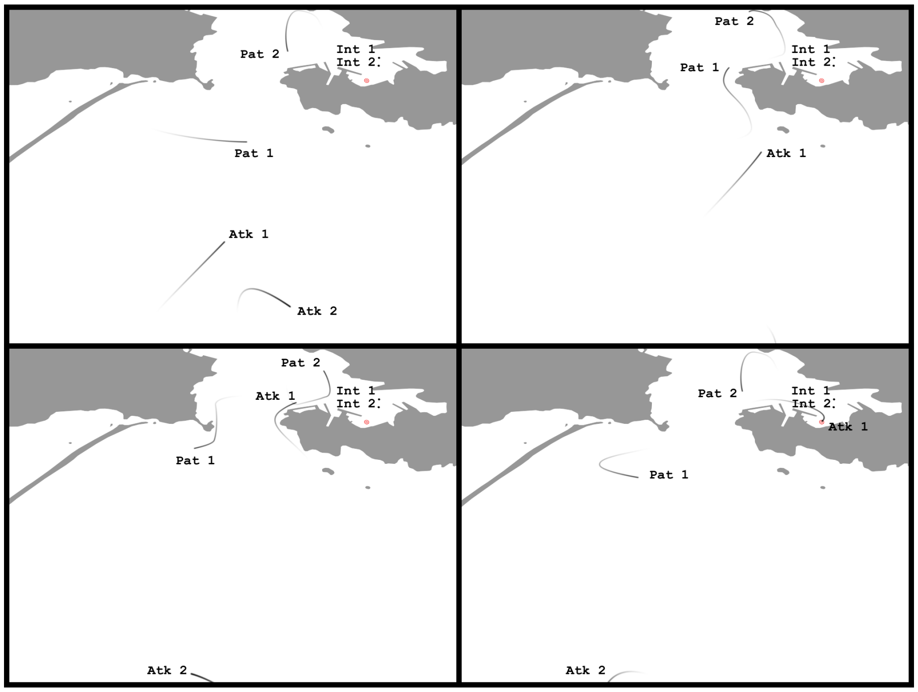

Figure 6: An attack scenario for $pat - int$ in Esquimalt found using $\rhd_{hsumafter}$

Attacker 2 again plays no significant role in the attack, but, since we are using the $f_{hidesum}$-variant in the goal ordering structure, it serves as a way to counterbalance the effects that Attacker 1 has on $f_{hidesum}$. After Attacker 1 followed the coastline and was detected by Patroller 1 in the second frame, Interceptor 2 was dispatched to try and stop Attacker 1. However, Attacker 1 accelerates, and by the time Interceptor 2 arrives, Attacker 1 has moved far beyond Interceptor 1's sensor range as seen in the third frame. Attacker 1 then makes its way to the target avoiding being detected by Patroller 2, and the dormant interceptors, according to policy, do not look out for it.

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

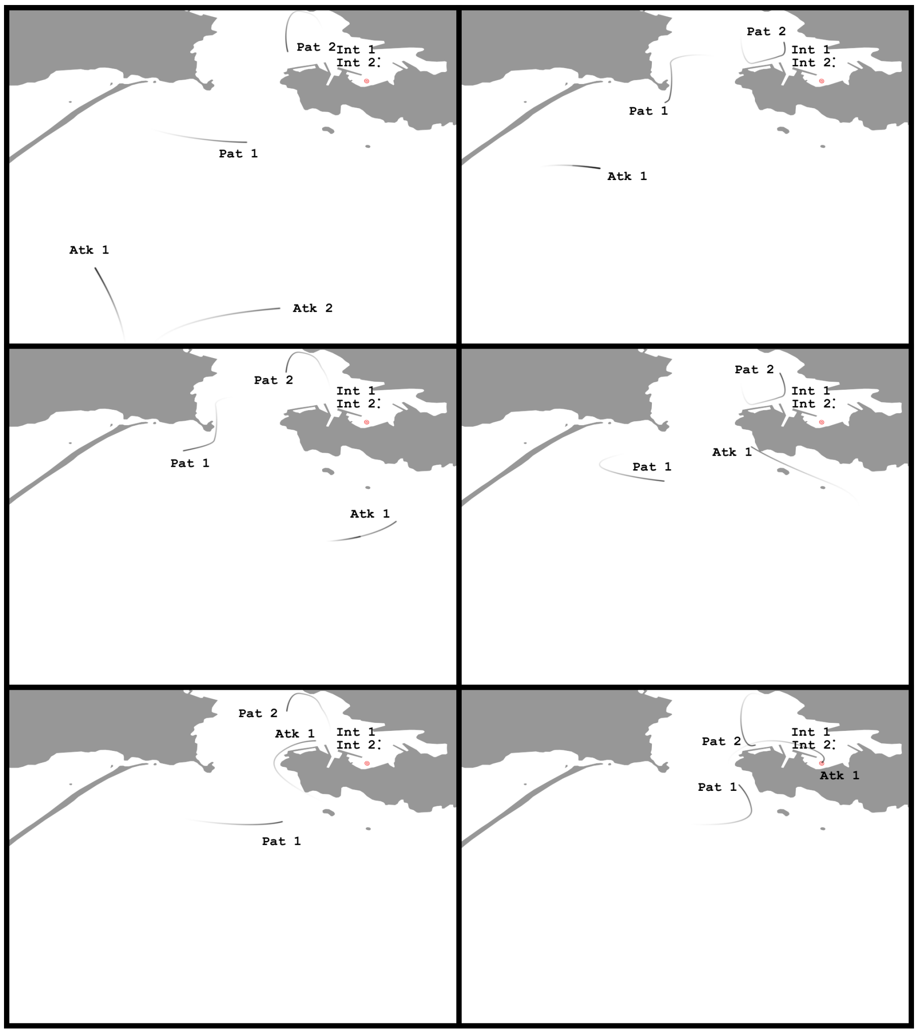

Figure 7: An attack scenario for $pat - int$ in Esquimalt found using $\triangleright_{hidebefore}$

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 already a representant of this general pattern (i.e. one attacker has entered the harbour and has

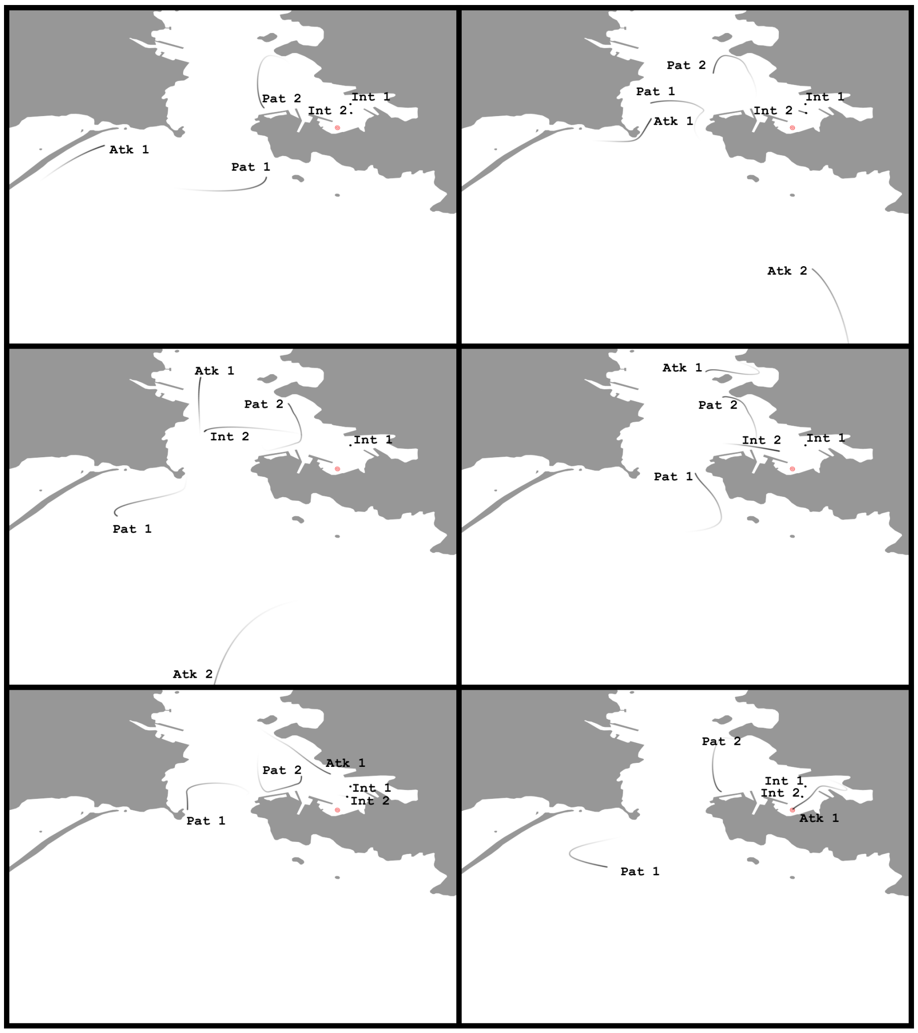

Figure 8: An attack scenario for $pat - int$ in Esquimalt found using $\triangleright_{hsumbefore}$

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, but, since such a pattern is also not too uncommon, it also explains why we have success from time to time.

Another surprising result from our result tables was that there was not so 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 the $f_{hide,i}$s 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". In Figure 5 we see that Attacker 2 does a little bit of a loop in approaching the harbour (Attacker 1 shows that there is a more direct way before turing away) creating more environment states where it stays far away from defenders before getting near them (which is done rather directly and quickly). Figure 7 shows this accumulation of hiding credit even more clearly by the zig-zag course of Attacker 1 essentially first going far left and then swing to the far right before entering the harbour.

All the attack behaviors 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 setup tests, which, again, is very important for testing.

Figures 9 and 10 show two more successful attacks that highlight a feature of our testing system and a little problem with our implementation of the policies. In Figure 9, we have another of the successful runs of our system using $\rhd_{base}$. Attacker 1 has a waypoint that is just shy of the shoreline, and since it was traveling at full throttle, it was not able to turn fast enough to avoid collision with the land. Despite this, Attacker 2 is still able to find the right timing to slip between Patrollers 1 and 2 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 exactly care much. 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 (withour even knowing what emergencies are).

In Figure 10, Attacker 2 is initially spotted by Patroller 1 in the second frame, and as such Interceptor 1 is dispatched. However, Attacker 2 comes close enough to Patroller 1 to cause the patroller to perform the interception; as such Interceptor 1 is instructed to return to its initial position. Before it can make it all the way back however, Patroller 1 spots Attacker 1, and

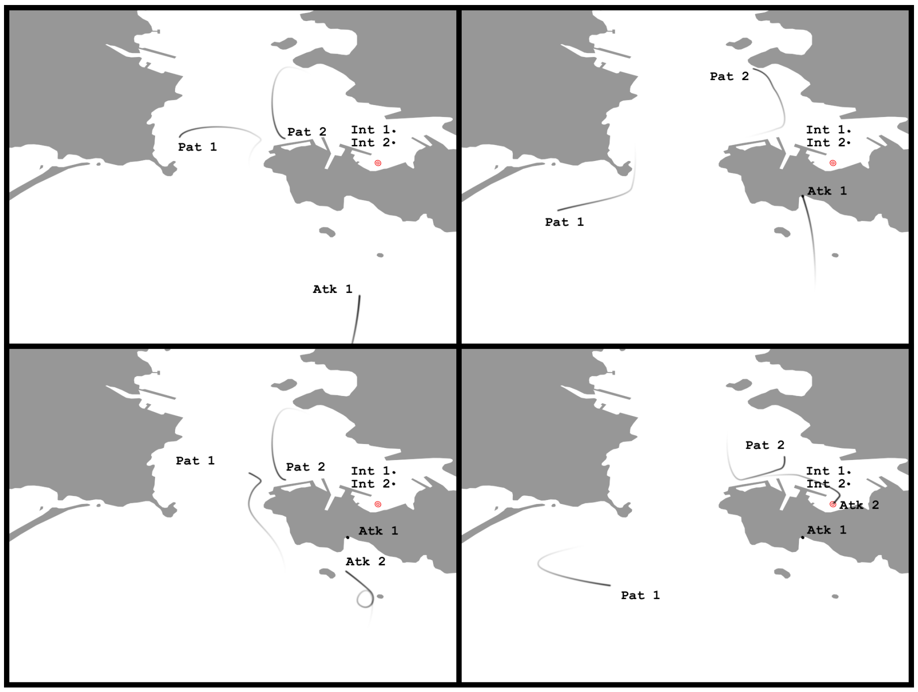

Figure 9: An attack scenario for $pat - int$ in Esquimalt found using $\triangleright_{base}$

instructs Interceptor 2 to intercept. Due to a implementation error (no collision avoidance between the defenders), the positions of the boats cause Interceptor 1 to collide with Patroller 2 on its way out to make the interception. According to policy, now Interceptor 2 takes the role of Patroller 2, and begins to follow the patrol route. Attacker 1 can then make it to the target without being intercepted, even though it is spotted by Interceptor 2 (which is now a patroller, so it will not move to intercept). 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.

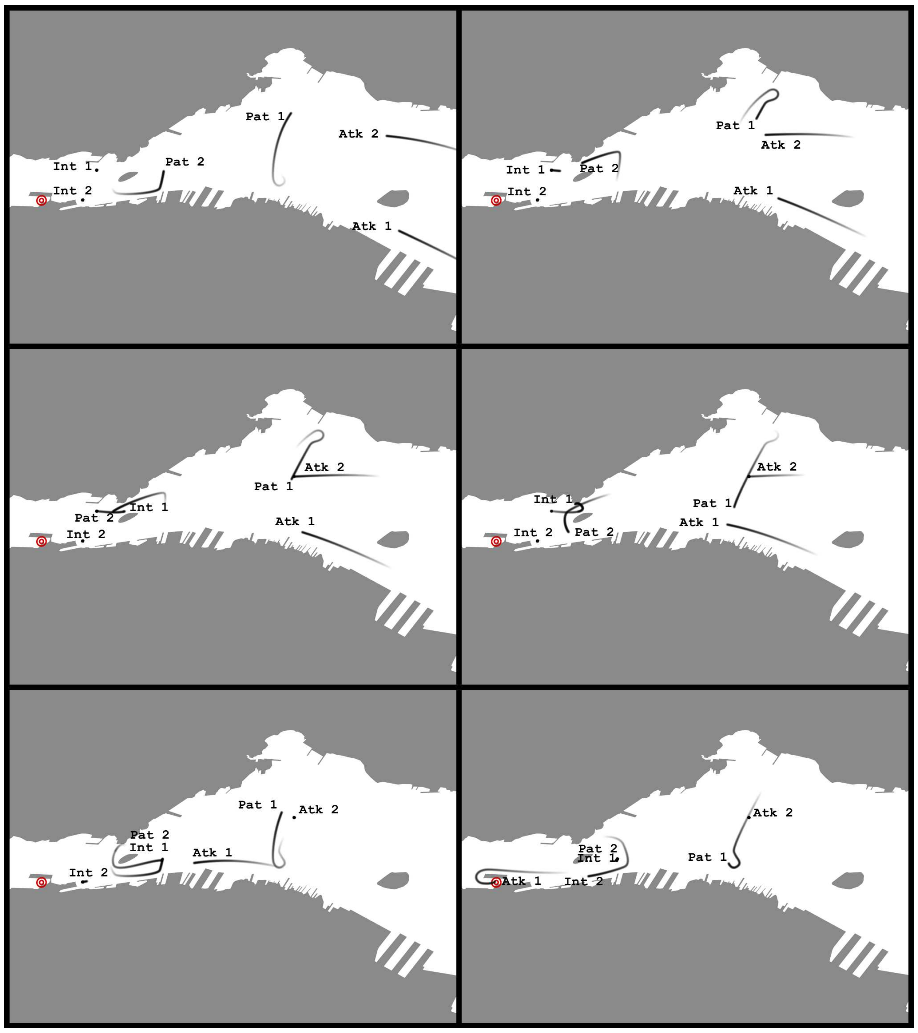

Figure 10: An attack scenario for $pat-int$ in Halifax found using $\rhd_{hidebefore}$

5 Related work

Using simulations for decision support, including testing all kinds of policies, is the topic of many papers, too many to even provide an overview (which would be its own paper). Even if we limit ourselves to multi-agent based simulations, nearly every application is with regard to decision support, like, for example, [10] in the area of financial market decision making, [1] in the area of tax evasion, or [19] in the area of traffic. But, as stated in the introduction, the testing of decision policies via simulations is done by humans in all of these applications.

[13] presents an approach that tries to automate the testing by having a human supply some typical behaviors and then mutate these behaviors automatically to achieve some more testing coverage. Our method does not need human guidance in form of typical behaviors (in fact, this obviously introduces a clear bias which is not very good with regard to testing) and it also uses a more sophisticated learning method than just mutations. But [13] is an example for the works in the area of search-based software testing, more specifically model-driven testing within this larger area, that has a large overlap with multi-agent based simulations and testing policies in them (see [8] for an overview of this area and [22] for a large, although incomplete, collection of papers around search based Software Engineering, including a lot of references on search based testing). If the models are executable, then they can be seen as simulations. Another example from this area is [3], which is not really a multi-agent based simulation, but finds task combinations that overstress a (simulated) scheduler. None of these approaches use PSS and none has something equivalent to our goal ordering structures.

If the goal is not to find weaknesses of a policy or concept, but ways how the policy or concept can be made to work (which can be seen as the opposite side of the coin to testing) then there are two approaches involving path planning in the literature. In [14], evolutionary methods are used to produce a configuration of an aircraft for a mission. The mission is then simulated using mostly a conventional path planner, which is similar to how we create the low-level waypoints. [21] presents an online learning system based on ant systems that tries to learn how to pass by defenders to reach a mission target. Naturally, online learning has the risk of bad early decisions that can make it impossible for the system to solve the given problem despite the fact that a solution exists. In both papers, the defenders are stationary, which obviously makes the problems easier than what we deal with. [12] presents the idea of using learning to, in our terminology, create a policy. Compared to testing, this is, in our opinion, a much more complex tasks, since instead of having to find *one* attack strategy, a policy should be able to work against *all* attacks. As such, the reported brittleness, compared to the robustness of our approach, by the authors of [12] is not surprising.

6 Conclusion and future work

We investigated different goal ordering structures for PSS-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 patrol and interception 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. But we are also interested to find ways to transfer the concept of goal ordering structures to other evolutionary learning methods that at the moment only allow single goal functions or at most a group of goal functions that are considered to be multiple objectives for the search.

References

[1] L. Antunes, J. Balsa, and H. Coelho: Tax Compliance Through MABS: The Case of Indirect Taxes, MASTA 2007, Guimaraees, 2007, pp. 605–617.

[2] A. Baldwin, M.C. Mont, and S. Shiu: Using Modelling and Simulation for Policy Decision Support in Identity Management, Proc. IEEE International Symposium on Policies for Distributed Systems and Networks, London, 2009, pp. 17-24.

[3] L. Briand, Y. Labiche and M. Shousha: Using Genetic Algorithms for Early Schedulability Analysis and Stress Testing in Real-Time Systems, Genetic Programming and Evolvable Machines 7(2), 2006, pp. 145–170.

[4] E.D. de Jong, R.A. Watson, and J.B. Pollack: Reducing Bloat and Promoting Diversity using Multi-Objective Methods, Proc. GECCO-01, San Francisco, 2001, pp. 11–18.

[5] B. Chan, J. Denzinger, D. Gates, K. Loose, and J. Buchanan: Evolutionary behavior testing of commercial computer games, Proc. CEC 2004, Portland, 2004, pp. 125–132.

[6] J. Denzinger and A. Schur: On Customizing Evolutionary Learning of Agent Behavior, Proc. 17th AI, London, ON, 2004, pp. 146–160.

[7] T. Flanagan, C. Thornton, and J. Denzinger: Testing harbour patrol and interception policies using particle-swarm-based learning of behavior, Proc. CISDA-09, Ottawa, 2009, on CD.

[8] M. Harman: The Current State and Future of Search Based Software Engineering, Proc. 29th ICSE: FoSE, Minneapolis, 2007, pp. 342–357.

[9] P.E. Hart, N.J. Nilsson, and B. Raphael: A Formal Basis for the Heuristic Determination of Minimum Cost Paths, IEEE Trans. Systems Science and Cybernetics 4(2), 1968, pp. 100–107.

[10] K. Izumi, H. Matsui, and Y. Matsuo: Socially embedded multi agent based simulation of financial market, Proc. AAMAS-07, Honolulu, 2007, p. 175.

[11] J. Kennedy and R.C. Eberhart: Particle swarm optimization, Proc. IEEE ICNN 1995, Piscataway, 1995, pp. 1942–1948.

[12] F. Klügl, R. Hatko, and M.V. Butz: Agent Learning Instead of Behavior Implementation for Simulations - A Case Study Using Classifier Systems, Proc. MATES'08, Kaiserslautern, 2008, pp. 111–122.

[13] E. Martin and T. Xie: A fault model and mutation testing of access control policies, Proc. 16th WWW, Banff, 2007, pp. 667–676.

[14] C. Miles and S.J. Louis: Case-Injection Improves Response Time for a Real-Time Strategy Game, Proc CIG-05, Colchester, 2005, pp. 149–156.

[15] T.E. Mora, A.B. Sesay, J. Denzinger, H. Golshan, G. Poissant, and C. Konecnik: Fuel Optimization using biologically-inspired Computational Models, Proc. IPC 2008, Calgary, 2008 (on CD).

[16] National Resources Canada: GeoGratis - Home, http://geogratis.cgdi.gc.ca/geogratis/en/index.html, as seen on Oct. 21, 2009.

[17] P. Parent and R. Church: Evolution of Geographical Information Systems as Decision Making Tools, Proc. GIS '87, Falls Church, 1987, pp. 63–71.

[18] D.J. Power and R. Sharda: Model-Driven DSS: Concepts and Research Directions, Decision Support Systems 43(3), 2007, pp. 1044–1061.

[19] M. Radecky and P. Gajdos: Intelligent agents for traffic simulation, Proc. SpringSim'08, New York, 2008, pp. 109-115.

[20] M. Reyes-Sierra and C.A. Coello Coello: Multi-Objective Particle Swarm Optimizers: A Survey of the State-of-the-Art, Int. Jour. Comp. Int. Res. 2(3), 2006, pp. 287–308.

[21] J.A. Sauter, R. Matthews, H. Van Dyke Parunak, and S. Brueckner: Evolving adaptive pheromone path planning mechanisms, Proc. AAMAS-02, Bologna, 2002, pp. 434–440.

[22] SEBASE: Software Engineering By Automated SEarch Repository, http://www.sebase.org/sbse/publications/, as seen on Oct. 21, 2009.