Tailoring a Cognitive Model for Situation Awareness using Machine Learning

Customer

National Aerospace Laboratory NLR

NLR-TP-2015-002 - February 2015

NLR – Dedicated to innovation in aerospace

National Aerospace Laboratory NLR
Anthony Fokkerweg 2
1059 CM Amsterdam
The Netherlands
Tel +31 (0)88 511 3113
www.nlr.nl

EXECUTIVE SUMMARY

Tailoring a Cognitive Model for Situation Awareness using Machine Learning

Problem area

In addition to training fighter pilots in a 'live' environment, it can be cost-effective to use some of these training hours within a simulated tactical environment, thereby fighting with and against Computer Generated Forces (CGFs). Such a training scenario will typically allow for the interaction between CGFs and human pilots. Behaviour of these CGFs may be accomplished by using a cognitive model. However, finding the most effective one is an expensive task, given that this typically entails testing of many models, thereby requiring lengthy sessions with different experts. It may therefore be preferable to automate this process, such that CGFs learn the most effective models by themselves.

Description of work

Constructing a self-learning CGF requires Machine Learning (ML), which is a branch of artificial intelligence. One type of ML is that of evolutionary computing; a group of optimization techniques that follow the theory of natural evolution as originally described by Charles Darwin (1809-1882). These techniques are particularly well suited for learning agent behaviour in an open environment. To this end, an evolutionary algorithm will be developed, of which its applicability will be determined on the task of finding the most effective cognitive models for simulated fighter pilots.

Report no.
NLR-TP-2015-002

Author(s)
R. Koopmanschap
M. Hoogendoorn
J.J.M. Roessingh

Report classification
UNCLASSIFIED

Date
February 2015

Knowledge area(s)
Training, Mission Simulation and Operator Performance

Descriptor(s)
Cognitive models
Neuroevolution
Machine Learning
Computer Generated Forces
Air Combat

This report is based on a paper accepted for publication in the special issue on ‘advances in applied artificial intelligence’ of the Springer journal ‘Applied Intelligence’.

Results and conclusions

The developed evolutionary algorithm and several extensions are evaluated with respect to performance in air combat. The results show that it is possible to apply the algorithm to optimize belief networks for cognitive models of intelligent agents (adversarial fighters) in the aforementioned domain, thereby reducing the effort required to elicit knowledge from experts, while retaining the required 'human-like' behavior.

Applicability

The technique combines the best of two worlds: it strives to results in human-like behaviour with a decreased need to consult domain experts. In order to further improve the learning behavior, several extensions have been proposed, of which some have shown to be successful in the current application. The work presented here, builds forth on earlier work performed within the Smart Bandits project (2009-2013) for the Royal Netherlands Air Force, but provides significant augmentations. In particular, the approach presented here requires only valid scenarios, whereas the technique developed within Smart Bandits requires domain experts to manually dictate the precise outcomes of the model being learned.

A key property of machine learning is the generalizability of its methods such that they are applicable to a wide variety of data. Hereto, an optimal model is typically treated as the solution to a numerical-optimization problem. Therefore, such a developed technique may be adapted to other problems with relatively little effort, such as learning (behavioural) models for more complex scenarios (four-versus-four, eight-versus-eight, etc.) with different fighter aircraft (F-16, F-35, etc.), different armament (Active, Semi-Active and Passive type of missile guidance systems, etc.), or different mission types (Offensive Counter Air, Defensive Counter Air). This may even be extended to other domains, such as autonomous manoeuvring of Remotely Piloted Aircraft.

Tailoring a Cognitive Model for Situation Awareness using Machine Learning

R. Koopmanschap[1], M. Hoogendoorn[1] and J.J.M. Roessingh

[1] VU University Amsterdam

Customer
National Aerospace Laboratory NLR
February 2015

This report is based on a paper accepted for publication in the special issue on 'advances in applied artificial intelligence' of the Springer journal 'Applied Intelligence'.

Customer	National Aerospace Laboratory NLR
Contract number	-----
Owner	National Aerospace Laboratory NLR
Division NLR	Air Transport
Distribution	Unlimited
Classification of title	Unclassified
Date	February 2015

Approved by:

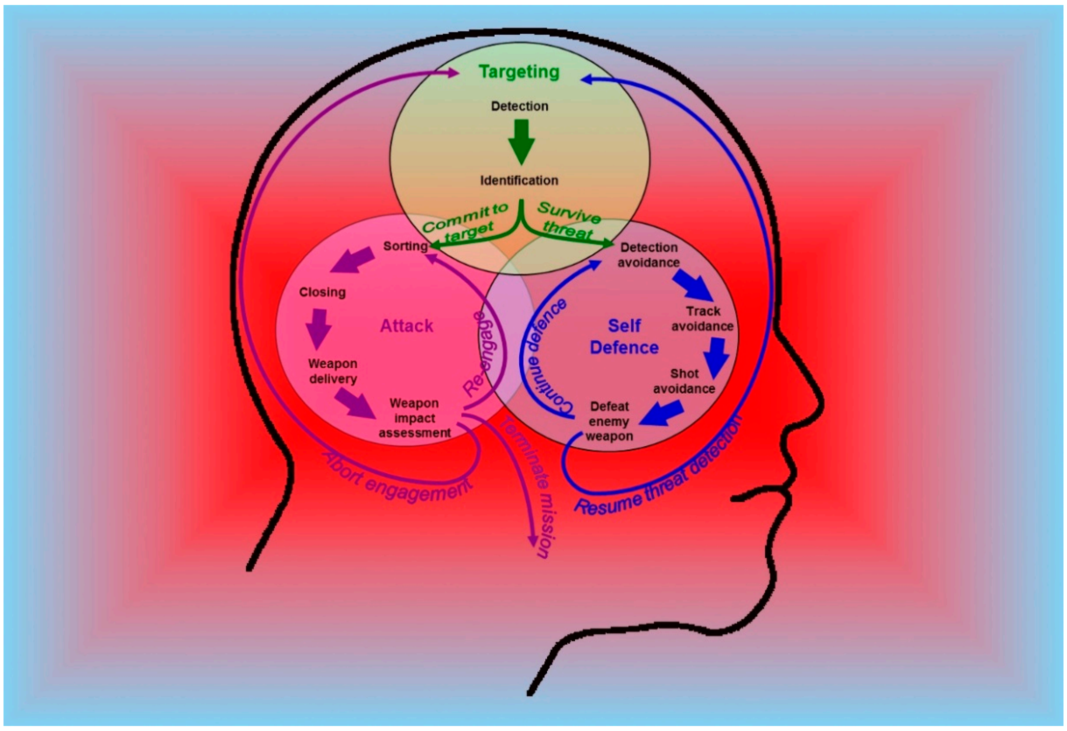
Author	Reviewer	Managing department
J.J.M. Roessingh [illegible]	R. Meiland [illegible]	H.G.M. Bohnen [illegible]
Date: 19/02/'15	Date: 19/02/'15	Date: 23/02/2015

Summary

Using a pure machine learning approach to enable the generation of behavior for agents in serious gaming applications can be problematic, because such applications often require human-like[1] behavior for agents that interact with human players. Such human-like behavior is not guaranteed with e.g. basic reinforcement learning schemes. Cognitive models can be very useful to establish human-like behavior in an agent. However, they require ample domain knowledge that might be difficult to obtain. In this paper, a cognitive model is taken as a basis, and the addition of scenario specific information is for a large part automated by means of machine learning techniques. The performance of the approach of automatically adding scenario specific information is rigorously evaluated using a case study in the domain of fighter air combat. An evolutionary algorithm is proposed for automatically tailoring a cognitive model for situation awareness of fighter pilots. The standard algorithm and several extensions are evaluated with respect to performance in air combat. The results show that it is possible to apply the algorithm to optimize belief networks for cognitive models of intelligent agents (adversarial fighters) in the aforementioned domain, thereby reducing the effort required to elicit knowledge from experts, while retaining the required 'human-like' behavior.

This paper is a significantly extended version of the following paper: Koopmanschap, R. Hoogendoorn, M., Roessingh, J.J., Learning Parameters for a Cognitive Model on Situation Awareness. Ali, M., Bosse, T., Hindriks. K.V., Hoogendoorn, M., Jonker, C.M., and Treur, J. (eds.), Recent Trends in Applied Artificial Intelligence, Proceedings of the 26th International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems, IEA-AIE 2013, Springer, LNCS 7906, 2013, pp. 22-32.

[1] Human-like behaviour is not the behaviour of the optimal performing and purely rational opponent. It is the behaviour of the agent with human shortcomings which is affected by such factors as fatigue, workload and constraints in situation awareness.


Targeting
Detection
Identification
Commit to target
Survive threat
Sorting
Closing
Attack
Weapon delivery
Weapon impact assessment
Re-engage
Detection avoidance
Track avoidance
Self Defence
Shot avoidance
Defeat enemy weapon
Continue defence
Abort engagement
Terminate mission
Resume threat detection

Content

Abbreviations

Acronym	Description
1V1	one versus one
ACT	Adaptive Control of Thought
ART	Adaptive Resonance Theory
CGFs	Computer Generated Forces
CLARION	Connectionist Learning with Adaptive Rule Induction On-line
DARPA	Defense Advanced Research Projects Agency
DM	Decision Making
EPIC	Executive-Process/Interactive Control
FLOT	Forward Line of Own Troops
FSM	Finite State Machine
ML	Machine Learning
NEAT	NeuroEvolution of Augmented Topologies
NLR	National Aerospace Laboratory NLR
OLLE	Off-Line Learning Environment
R&D	Research and Development
RWR	Radar Warning Receiver
SA	Situation Awareness

1 Introduction

In many settings, it is highly desirable that agents exhibit human-like behavior. For example, in the serious gaming domain having human-like computerized teammates or opponents can contribute to a more realistic and therefore potentially more effective training experience [1]. For instance, for the domain of fighter pilot training, in order for trainees to learn appropriate tactical avoidance maneuvers in a simulator, it is essential to include an opponent fighter aircraft in the simulation that attacks in a way a human pilot would. Essentially, two basic approaches can be contrasted to establish such behavior:

(1) a knowledge based approach in which the agents are equipped with cognitive models that replicate human behavior and are equipped with expert knowledge

(2) a mere machine learning perspective in which the behavior is learnt by means of examples or by learning appropriate behavior using simulations.

As there is no guarantee that using machine learning results in human behavior, quite some attention has been devoted to the development of cognitive models to facilitate the generation of this desired human-like behavior, for instance models for Situation Awareness (SA, see e.g. [2]), theory of mind (see e.g. [3]), naturalistic decision making (e.g. [4]), and many more. However, such models require a substantial amount of domain knowledge. In an SA model, for example, knowledge is needed on how to interpret certain observations for the domain at hand. In a decision making model, knowledge is needed about the (relative) suitability of certain decision options, given the model's inputs. An approach to specify this knowledge is to involve domain experts, extract their knowledge, and incorporate this knowledge in the cognitive models. However, this requires a substantial effort from experts which are most often scarcely available and expensive in terms of time. To deal with this disadvantage, a combination of cognitive modeling and machine learning might be able to provide a solution: the cognitive model ensures that only human-like behavior can be generated as it is based on theories of human reasoning and the machine learning component learns part of the knowledge required for the scenario at hand. This could result in a reduction of effort to elicit knowledge from experts, while maintaining the human-like behavior.

One example of an approach which combines a cognitive model with learning is presented in Gini, Hoogendoorn and van Lambalgen [5], who applied learning to an SA model. Gini et al assume that knowledge about links between observations and judgments of the situation is provided by domain experts. Furthermore, the proper functioning of the SA model is based on the assumption that experts provide a correct judgment for all situations which the agent might encounter (i.e. the judgment of the outcome for all possible sets of observations). Subsequently, the learning algorithm finds appropriate values for the strengths of connections between observations and the judgment of the situations. Although the results are promising and the approach reduces a substantial part of the burden for the domain expert, it still requires the domain expert to go through a large number of hypothetical situations and to explicitly specify the possible outcomes of each situation.

In the current paper, we substantially extend the approach described by Gini et al, thereby still taking the developed cognitive model (which is claimed to exhibit human-like behavior, see [2]) as a basis. Instead of assuming the desired output to be provided by an expert and learn the connection strengths based on this output, learning is based on the performance of the agent in scenarios, specifically designed to train SA. We used and refined various machine learning algorithms to maximize the learning ability of the approach. We applied this approach to air-to-air combat of fighter pilots and systematically evaluated the learning algorithms for a large number of scenarios. The findings support our claim that this approach will reduce the effort required to elicit knowledge from experts while retaining human-like behavior. The overall approach is novel through its combination of a cognitive model of situation awareness with scenario-based learning without the need for domain expertise.

This paper is organized as follows. In Section 2 the background is explained. Section 3 presents the approach that is used to learn the domain knowledge encapsulated in the SA model. A case study is presented in Section 4 whereas Section 5 discusses the details of the implementation. The results of the application of the machine learning approach to the case study are presented in Section 6. Finally, Section 7 provides a discussion.

2 Background

2.1 Cognitive Modelling

Cognitive models deal with the symbolic-information processing level and are thought to be largely independent of models at the physiological (neurological, millisecond) level. Moreover, [25] argues that these models will continue to be largely serial, ‘for the mind behaves like a serial system wherever the bottleneck of attention supervenes upon events’. Some of those models are based on unified theories of cognition. The goal of such a model is to show how a single control structure can handle all of the cognitive processes of which the human mind is capable. Soar [12], ACT-R [13], EPIC [14] and CLARION [15], are frequently cited in connection with this class of models. In both Soar and ACT-R, cognition is largely synonymous with problem solving. The Soar and ACT-R architectures can be seen as similar in several ways. They are based on production systems (basically if .. then ..-rule systems) that require two types of memory: ‘declarative’ memory for facts and ‘procedural’ memory for rules. One significant initiative in the air combat

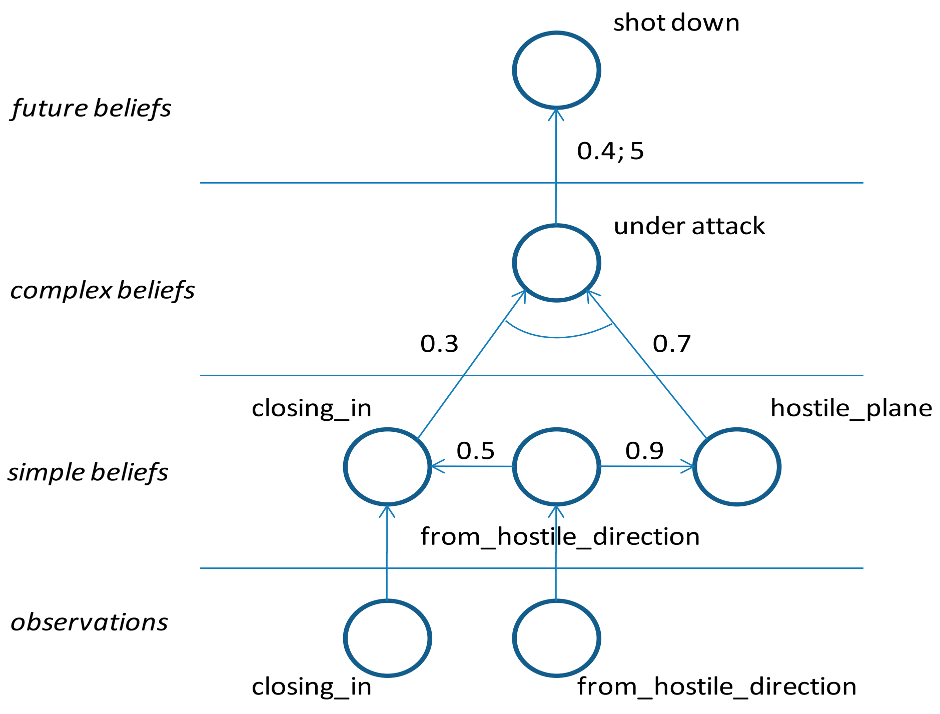


Fig.1 Example belief network model ‘hostile aircraft’ (cf. [5])

domain, initially funded by the US Defense Advanced Research Projects Agency (DARPA) is TacAir-Soar [21]. TacAir-Soar is based on the Soar architecture and generates believable human-like behavior for large-scale, distributed air combat simulations. TacAir-Soar is capable of controlling many agents simultaneously in a large range of missions by integrating a wide variety of intelligent capabilities, including real-time hierarchical execution of complex goals and plans, communication and coordination with humans and simulated entities, maintenance of situational awareness and the ability to accept and respond to new orders while in flight.

In contrast, the current research merely aims at developing agents that play the role of adversaries in restricted mission types, so called air-to-air missions and more specifically Beyond Visual Range counter-air missions, in which the virtual adversary aircraft remain outside the ‘visual range’ of their opponents. The tactical behavior of these adversaries is on the ‘formation level’, rather than on the level of a large scale air campaign. The behavior of aircraft can only be observed by the on-board sensors of their opponent, and vice versa. While the innovative character of TacAir-Soar was primarily a matter of scale and integration, the focus of the current research is to evaluate the use of machine learning techniques to implant scenario-specific domain knowledge in the artificial brain of virtual adversaries.

The current research focuses on weight learning in belief networks, rather than weight learning in production rules. While TacAir-Soar did not use the built-in learning abilities of Soar [21], a weight learning technique in the rule-based category may turn to be effective for the learning of human-like behavior. An example of a straightforward technique, developed in the context of game AI, is ‘dynamic scripting’ [22]. This is a reinforcement learning technique, in which subsequent versions of behavioral scripts for an agent are based on increasingly successful rules. After each turn, engagement, or trial of the game, the weights of the rules in the script will be updated on the basis of a reward function, depending on the success of the script. The rules will then be put back in a rule base, from which the rules of a subsequent version of the script are drawn. Rules with higher weights thus have an increased probability to become selected. This technique has been successfully applied to air combat simulation by Toubman et al [23]. However, the learning technique and the superior tactical behavior of agents based on dynamic scripting are not based on cognitive models, such that behavior of the agent is not constrained to be human-like.

The current research uses component cognitive models, rather than large-scale integrated models based on cognitive architectures such as Soar. Such component models have been created to facilitate the generation of certain aspects of human-like behavior of role-playing agents in simulations for complex skill training. Examples are models for decision making that may be biased by ‘intuition’ or emotions such as fear [4], and for behavior that is influenced by being surprised [20].

Such a component cognitive model for Situation Awareness, which is subject to learning, is a belief network proposed by Bosse et al [6], which is a slightly modified version of [2]. The model is meant to create high-level judgments of the current situation following a psychological model from Endsley [7]. The model includes three forms of beliefs, namely *simple beliefs*, which can be directly derived from observations performed by the agent, or other simple beliefs. Furthermore, *complex beliefs* express combinations of simple beliefs, and describe the current situation on an abstracted level, and finally, *future beliefs* express expectations of the agent regarding projected (future) events in the scenario. All these beliefs are assigned an activation value between 0 and 1 and are connected via a network, in which each connection between a pair of beliefs has a particular strength (with a value between -1 and 1). Several update properties are specified that express how new knowledge obtained through observations is propagated through the network using the activation value of beliefs as a rank ordering of priority (the most active beliefs are updated first). For the sake of brevity, the details of the updating algorithm have been omitted; see [6] for an in-depth treatment of the update rules. The domain knowledge that is part of the model is a *belief network*. An example of such a network is shown in Figure 1.

The example shows that each connection between a pair of beliefs receives a connection strength. For instance, the connection between the simple belief *from hostile direction* and the simple belief *hostile plane* has a connection strength of 0.9. This relatively high value will help to activate the belief that an approaching aircraft is indeed hostile when it is observed to come from a hostile direction. The connections between complex beliefs and future beliefs also have a delay parameter (the second number on the connection), which is five units of time in the example network, to specify at what time (relative to the current time) the agent believes that the future belief will occur. In larger, more meaningful, networks, the task of determining and assigning the appropriate values for connection strengths and future belief delays needs to be done by domain experts, which consider this task as cumbersome and prone to error. Therefore, this task will be allocated to a learning algorithm. Note however that the conceptual belief structure of the network must still be provided, a task for which expert input is still needed, but which is considered less cumbersome and error prone.

In order to enable learning based upon the performance of the agent in scenarios, the agent needs not only to judge the situation, but additionally needs to decide on an action. To this end, we have extended the SA model with a simple Decision Making (DM) model, namely a Finite State Machine (FSM) model. Other DM models, such as the more human-like DM developed in [4], could be applied here as well. However, since we were primarily interested in evaluating the learning of a component model (the SA model) a non-adaptive FSM was chosen to implement the subsequent DM. The functioning of the DM model is briefly explained in Section 4.3.

2.2 Machine Learning

A variety of machine learning techniques have been introduced to perform the learning proposed in this paper, namely to learn weights in a network. Essentially, techniques such as Simulated Annealing [24] or Genetic Algorithms [18] can be deployed. Of course, techniques from the domain of Neural Networks (NNs) are also obvious choices, as these techniques are also concerned with learning the weights in a network. Instead of simple algorithms such as backpropagation, also more sophisticated approaches such as NEAT [17] could be deployed. In addition to adjusting the weights in the network, NEAT also adjusts the topology of the network. However, the latter type of approach would not be considered necessary for the current purposes, since we assume that a suitable network (topology) can be distilled from domain experts. Hence, merely the weights of the connections should be learned. Provided that the network to be learned in this paper has some different characteristics compared to standard NNs, the choice has been made to focus on Genetic Algorithms to learn the weights[2].

[2] Reasons for this choice are that applying genetic algorithms to neuro-evolution is particularly popular and has been shown to be competitive with other neural network learning techniques, such as backpropagation [26] and it was asserted in [5] that a genetic algorithm is capable of reaching good results and eventually convergence to a near optimal solution for this type of problem.

3 Learning Approach

In this Section, the learning approach is presented. First, the basic algorithm is explained in detail, followed by a number of extensions to improve the learning process.

3.1 Basic Learning Algorithm

The agent's SA-based decision making model includes:

1. The structure of the SA belief network (i.e. the precise beliefs that should be present and the connections that exist between the beliefs) as well as the rules that underlie the calculations of new belief activation values (as explained in Section 2.1).An example of such a belief network is the one shown in Figure 1.
2. The full decision making model that maps activation values of beliefs to concrete actions for the specific domain at hand. For example, a decision making model that could be used with the SA belief network in Figure 1 could consist of a single rule stating that if the belief “under attack” has an activation value higher than 0.5, the action to flee / defend is initiated.

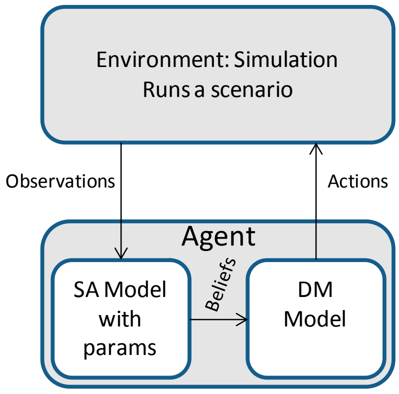


Fig 2 Overview of running a scenario

The outcome of the learning process will be connection strengths within the belief network. In order to learn these connection strengths in an appropriate manner, the following approach is proposed:

1. Domain experts provide a scenario that facilitates the learning of specific behavior. Hereby, the scenario allows for appropriate behavior and less appropriate or even unwanted behavior of the agent, and this can be measured by means of a performance metric (in this case called a fitness). A scenario example for the network in Figure 1

could be a hostile plane flying towards the agent and fires a missile that destroys the agent if it does not turn around fast enough.

2. Using the scenarios, we apply machine learning techniques that attempt to learn the best (i.e. most appropriate) behavior of the agents within the scenarios by finding the best settings for the connection strengths. In the belief network of Figure 1 the values that should be learned consist of all connection strengths shown, (i.e.: (0.4, 0.3, 0.7, 0.5, 0.9)).

In Figure 2, a high level overview of running a simulation for a certain scenario is shown given certain parameter settings for the situation awareness model. The simulation is the outside world that keeps track of the location of all objects and contains the rules of the world. The simulation contains a scenario that specifies the start conditions and precise settings of the world. The agent receives observations from the simulation and sends these to its situation awareness model. The model then generates a high level view with the observations about what it believes is happening in the world given the current parameter settings within the model. The high level view of the situation is transferred to the decision making model, which uses the activation values of selected beliefs to generate an action, which, in turn, is sent back to the simulation. Once a single run of the scenario has been completed with certain settings of the model parameters, a fitness value can be assigned to the agent, based on its performance in the scenario. Depending on the exact scenario that is run in the simulation, a different belief network with different parameters, is required.

Algorithm 1. *Basic learning loop*

```
#   Vector learn weights (max runs, scenario time, alg)                          Extension
1   initialize the connection strengths vector V uniform random                  (3),(4)
2   runs = 0;
3   while (runs <max_runs)
4       initialize the scenario                                                  (5)
5       while (time <scenario_time)
6           determine the current observations
7           feed the observations used as input for the SA model to the agent
            controller by the SA model
8           determine activation values of the beliefs with V (SA model)
9           derive the actions of the agent with the activation values of beliefs
            (DM model)
10          perform the actions in the world                                     (1)
11          time++
12      End
13      determine the performance in the scenario (fitness value)                (2)
14  if (alg == "evolutionary")
15      if all individuals in the population have been evaluated
16          create a new population using variation and selection operators and
            select the new V from the new population                             (3), (4)
17      else
18          select a V in the current population that has not been evaluated yet
19      if (alg == "hill climbing")
20          generate a new V using hill climbing                                 (3),(4)
21      runs++
22  End
23  if (alg == "evolutionary") return the best V in the population
24  if (alg == "hill climbing") return V
```

Given the ability to run such a scenario, a learning loop can be identified and is expressed in Algorithm 1 addressing the learning of the weights (note that the position where extensions will be made as described in Section 3.2 are indicated in the pseudocode). Two different learning algorithms are tested and extended with several methods to enhance their performance: a simple hill climbing algorithm and an evolutionary algorithm. Both are shown in the algorithm description. All algorithms represent the search space as a vector of all weights in the network. For the evolutionary algorithm the chromosome specifies these weights in sequence, for instance the weights as shown in Figure 1 are represented as follows:

0.4	0.3	0.7	0.5	0.9

Both the hill climbing and evolutionary algorithm use Gaussian mutation to generate offspring. Each scalar in the vector has a predetermined mutation probability and a sigma with which the Gaussian mutation performed. In Gaussian mutation, a Gaussian distribution is used to generate a mutated value for a number in the weight vector. The sigma indicates the standard deviation of the distribution. The evolutionary algorithm applies tournament selection for both the survivor selection and the parent selection. For crossover a uniform crossover operator is used. Note that not all these parameters have been mentioned in the pseudocode to avoid too much detail.

3.2 Learning Algorithm Extensions

Since the learning process takes place in a simulated environment, performance issues will quickly emerge. Therefore, five extensions to the above algorithms have been developed to speed up the learning process. These extensions can be divided into two categories: extensions that alter the fitness function (extension 1 and 2) and extensions that limit the search space (extension, 3, 4, and 5). The following extensions are applied to the algorithms. Note that the location where the extensions are placed in the algorithm itself are shown in the description of Algorithm 1.

(1) Adding output noise. In order to let the decision making model make a decision, the activation value of the required beliefs needs to exceed a certain threshold. This results in the effect that small changes in connection strengths between beliefs in the SA model are unlikely to cause beliefs to exceed the threshold and thus, do not result in any differences in fitness. Machine learning algorithms have difficulty finding an optimum in such cases. In order to deal with this problem, an approach has been developed: a random bias is assigned to the activation value of each belief at the start of each trial of a scenario. This value is taken from a Gaussian distribution with parameters that can be set to an appropriate value. This random bias is added to the activation value of a belief whenever the decision making model uses the value of the belief. This may cause the activation value to pass the threshold (in either direction) when it otherwise would not have. The frequency of passing the threshold depends on the difference between the activation value of the belief in the scenario and the threshold in the decision making model. The decision surface for the learning algorithm would presumably become smoother if this process is repeated a sufficient number of times for each connection strength vector. This would occur because the noise will provide an averaged fitness of the values around the activation value currently achieved, and would therefore show an averaged performance in the region of the current activation value. The extension is expressed in Algorithm 2 concerning the learning of weights with output noise, whereby the changes are shown in bold italics.

***Algorithm 2.** Learning loop with output noise*

```
Vector learn_weights_output_noise(max_runs, scenario_time, alg)
initialize the connection strengths vector V uniform random
runs = 0;
while (runs <max_runs)
initialize the scenario
for(each belief x)
        generate a random number bias      (b(x) = gaussian_rand(σ,μ))
    while (time <scenario_time)
        determine the current observations
        feed the observations used as input for the SA model to the agent controller by the SA model
        determine activation values of the beliefs with V(SA model)
        derive the actions of the agent with the activation values of belief(x) + b(x) (DM model)
        perform the actions in the world
        time++

    end
    determine the performance in the scenario (fitness value)
if (alg == "evolutionary")
                if all individuals in the population have been evaluated
                        create a new population using variation and selection operators and
                        select the new V from the new population
                Else
                        select a V in the current population that has not been evaluated yet
    if (alg == "hill climbing")
                generate a new V using hill climbing
    runs++
End
if (alg == "evolutionary") return the best V in the population
if (alg == "hill climbing") return V
```

(2) Expanding the fitness function with scenario-specific information. For obvious reasons, it is preferable to use a fitness function that does not require a substantial amount of information from the domain. For instance, by just providing a payoff for a successful run in a scenario or giving a penalty for a non-successful one. However, sometimes a more fine-grained view is necessary to learn the finer details of the domain (for example, providing fitness for specific milestones reached). Therefore an option for an extension is also to provide such a more fine-grained fitness function. In Algorithm 1 this only influences the way in which the fitness is calculated (line 13).

(3) Using symmetry information. In conventional neural networks the precise meaning of the nodes is often difficult to determine. In cognitive models, on the other hand, the nodes are constructed for specific concepts that are often well understood. Although it may not be clear what the optimal connection strength between two beliefs is, it is often clear that many parts in a belief network are symmetrical. For example, the connection between beliefs that deal with situations to the North may be equivalent to similar connections between beliefs that deal with situations to the South. By explicitly defining such symmetries beforehand, the learning algorithm can limit the search space by always assigning these connections the same weights. An additional advantage of defining symmetries is that it allows the agent to generalize its learned behaviour to situations that have not been previously encountered. If the agent would never have seen a training example that activates the beliefs dealing with situations to the South, the agent may still be able to handle the situation correctly if it has seen similar examples to the North. In the algorithm it limits the initialization and updating possibilities of the Algorithm 1 (lines 1, 16 and 20).

(4) Limiting value range in the search space. Values in the search space of the fully learned belief networks of the SA model do not emerge equally often. In particular, negative values for connection strengths are less common. The efficiency of the learning algorithm can be improved by reducing the set of values with which it can update the weight vector. For example, only the negative value -1 could be allowed in addition to the value range 0-1. By removing uncommon values, unpromising areas of the search space can be removed before the learning starts. Again, this influences the possibilities in lines1, 16, and 20 of Algorithm 1.

Algorithm 3. Learning loop with incremental learning

```
    Vector learn_weights_incremental (max_runs, scenario_time, alg)
1   for (each scenario)
2           load all stored connection strengths
3           initialize the connection strengths vector with the still unknown connection strengths
4   runs = 0;
5   while (runs <max_runs)
6       initialize the scenario
7       while (time <scenario_time)
8           determine the current observations
9           feed the observations used as input for the SA model to the agent controller by the SA model
10          determine activation values of the beliefs with V(SA model)
11          derive the actions of the agent with the activation values of  beliefs (DM-model)
12          perform the actions in the world
13          time++
14      end
15      determine the performance in the scenario (fitness value)
16      if (alg == "evolutionary")
17          if all individuals in the population have been evaluated
18                  create a new population using variation and selection operators and select the new
                    V from the new population
19          else
20                  select a V in the current population that has not been evaluated yet
21      if (alg == "hill climbing")
22          generate a new V using hill climbing
23
24      runs++
25  end
26  if (alg == "evolutionary") select the best V in the population
27  if (alg == "hill climbing") select V
28  end
```

(5) Incremental Learning. It has been shown in previous research that training an agent on a simple task before training it on a more difficult task can improve results (see e.g. [8]). As the meaning of the nodes in the network are well understood, it is fairly straightforward to use this technique here to train parts of the network in scenarios that are specifically designed to learn those simple tasks. The learned connection strengths can then be stored and used for more complex tasks that require simpler parts of the network to work correctly. For the purpose of this research, a part of the complete belief network is first trained in a scenario. When the scenario is completed, all connection strengths of that part of the belief network are stored. The algorithm then starts the next scenario with a larger part of the belief network and loads the stored weights from the first scenario to re-use them in the current one. Any number of scenarios can be linked, provided that the belief networks contain more nodes each subsequent time. In the approach, the algorithm does not change connection strengths that it has already learned and so it is left to the designer of the scenarios to ensure that the connection strengths accurately represent the connections between beliefs after being trained in the first scenario. Algorithm 3 describes the algorithm with the use of the incremental learning extension.

4 Case Study and Experimental Setup

In order to test the suitability of the approach, it has been applied in the domain of air combat, which is an interesting domain as great efforts are made to find more effective ways of training (also from a cost perspective) than the current training approaches. Our main hypothesis is that the evolutionary approach will outperform other algorithms and the individual as well as combined extensions will contribute to a better overall performance. There is an interest from this domain in agents exhibiting realistic tactical behavior so as to increase the value of simulation training for fighter pilots. Although the focus lies on demonstrating adversarial behaviour in air-to-air missions, the results are more widely applicable in the simulation domain. First, some basic required background knowledge about the domain is provided. Then the case study scenario is described in more detail, followed by the specific situation awareness models and decision making models used. Finally, the experimental setup is discussed.

4.1 Air Combat Domain

Two important sensors of a typical fighter aircraft are the radar and the Radar Warning Receiver (RWR). The radar is able to detect other aircraft within a cone-shaped volume extending from the nose of the aircraft. In our study case, the radar may operate in one of the following three modes: (1) search, (2) track and (3) lock. In search mode, other aircraft can be searched and detected in a wide area. In track mode, the cone-shaped volume is considerably reduced, but movements of aircraft (targets) within the volume can be more accurately tracked. In lock mode, the radar is capable of detecting target movements precisely, which is required to fire a missile and guide it towards a target.

Whether or not a target can be detected by radar is dependent upon the angle that the target has in relation to the aircraft. If a target is flying perpendicular to the direction of the radar waves, it will be difficult to detect by radar. Therefore, a typical evasive maneuver that can be used to avoid being detected, tracked or locked is to initiate a maneuver perpendicular to the assumed direction of a hostile aircraft. This is referred to as 'beaming'.

The RWR is a sensor that picks up radar activity. If an aircraft equipped with RWR picks up radar activity, it is able to establish the general direction of that activity, though not the exact location of the source. Even if the aircraft with the RWR is outside the detection range of the emitting radar, the RWR may be able to pick up its radar activity.

Missiles can be fired when a target is locked. The missiles that are assumed in this case study have autonomous capability to track designated targets and navigate towards it. Thus, after missile launch, the attacker does no longer have to point its radar on the target. The target aircraft can either try to manoeuvre away from hostile missiles, or can try to become invisible for the hostile radar through the aforementioned beam manoeuvre.

A final concept of importance for the case studies is the direction that aircraft are coming from. If an aircraft is coming from the FLOT ('Forward Line of Own Troops') it is more likely to be hostile than aircraft coming from any other direction.

4.2 Scenarios

In this case, a 'one versus one' (1v1) complete engagement scenario has been selected, whereby a first agent using the described cognitive models is patrolling an area and encounters a second, scripted, agent. Currently, a two-dimensional view of the scenario is adopted (i.e. aircraft movements in the third dimension, in this case altitude, are not enabled). The scripted agent can either be hostile or friendly (the latter type must clearly not be attacked) and has fixed behaviour for both settings. Good behaviour for the cognitive agent is to first detect the other aircraft and identify it as either friendly or hostile (i.e. identification behaviour), based on where it is coming from and whether it responds to identification protocols. If the scripted agent is identified as hostile, the cognitive agent should avoid missile attacks from the scripted agent and attempt to destroy it.

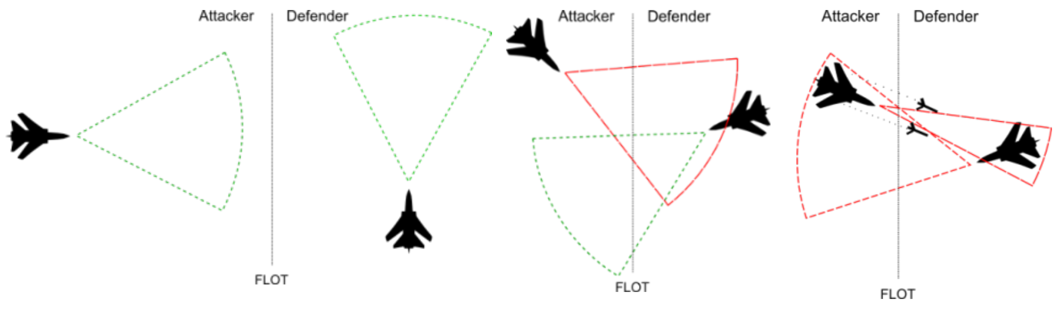


Fig. 3 Overview of 1v1 scenario (from left to right: unaware of presence, attacker detects defender, two missiles fired)

Each aircraft may perform any number of actions (see Section 4.3 for the precise actions), among which are manoeuvres that minimize the probability of detection on radar by the enemy, evasive manoeuvres, tracking and intercepting an opponent, making a radar lock on the opponent, and firing missiles on the opponent. In addition, an aircraft is free in its ability to roam around, provided that the movements are limited to the horizontal plane (to simplify matters, aircraft are unable to initiate vertical movements). When the opponent locks and fires a missile, the aircraft will attempt to defeat said missile by either trying to move outside of the missile range or by initiating the aforementioned beam manoeuvre. Figure 3 sketches the three main stages of such a scenario. First, both aircraft are on their own side of the FLOT, and are unaware of each other's presence (Left). Then, 'Attacker' detects the 'Defender' on its radar (Middle). After achieving a lock on ‘Defender’, ‘Attacker’ fires two missiles. The scenario will end with the destruction of an agent, or after a pre-set maximum amount of time has passed.

The full scenario is far from trivial and difficult to learn. Therefore, in order to be able to experiment with the various algorithms and extensions proposed, six simpler scenarios have been defined. These simpler scenarios cover all the behaviour necessary to perform the full scenario. For example, one of the simpler scenarios requires the agent to correctly understand when it has to fire a missile, another scenario requires the agent to know how to avoid enemy fire and so on. One of the scenarios does integrate multiple such aspects (but in a less complex way) and is referred to as the *awareness scenario*. Essentially, it deals with the identification and remembering of nearby aircraft. The latter scenario is used for most experiments, as it is neither too simple nor too complex. Therefore this scenario and its cognitive model will be described in greater detail.

In the awareness scenario, the agent is able to spot either a friendly or a hostile aircraft for a brief period. Afterwards the agent is forced to change its flight direction, causing it to lose its view on the target aircraft for some time. Subsequently, the agent can choose to turn back into the direction of the target. If the agent believes that the target is in that specific direction, it can quickly make a lock on it and fire at it. Since the agent was unable to continuously track the target, it had to remember its direction in order to be able to reacquire it, later. Thus, the scenario requires the agent to correctly identify the target and remember its direction in order to be able make the correct decision whether to later engage the target aircraft. In the scenario, the speed is controlled by the agent and the initial position is selected from a limited set of possible starting locations.

4.3 Cognitive Models

In order to learn the appropriate behaviour for the scenarios, a belief network (as part of the SA model) has been developed that can capture the behaviour required. In this case the network for the awareness scenario is presented. This belief network also encompasses certain aspects that can also be trained in the simpler identification behaviour scenario. Figure 4 shows the developed belief network.

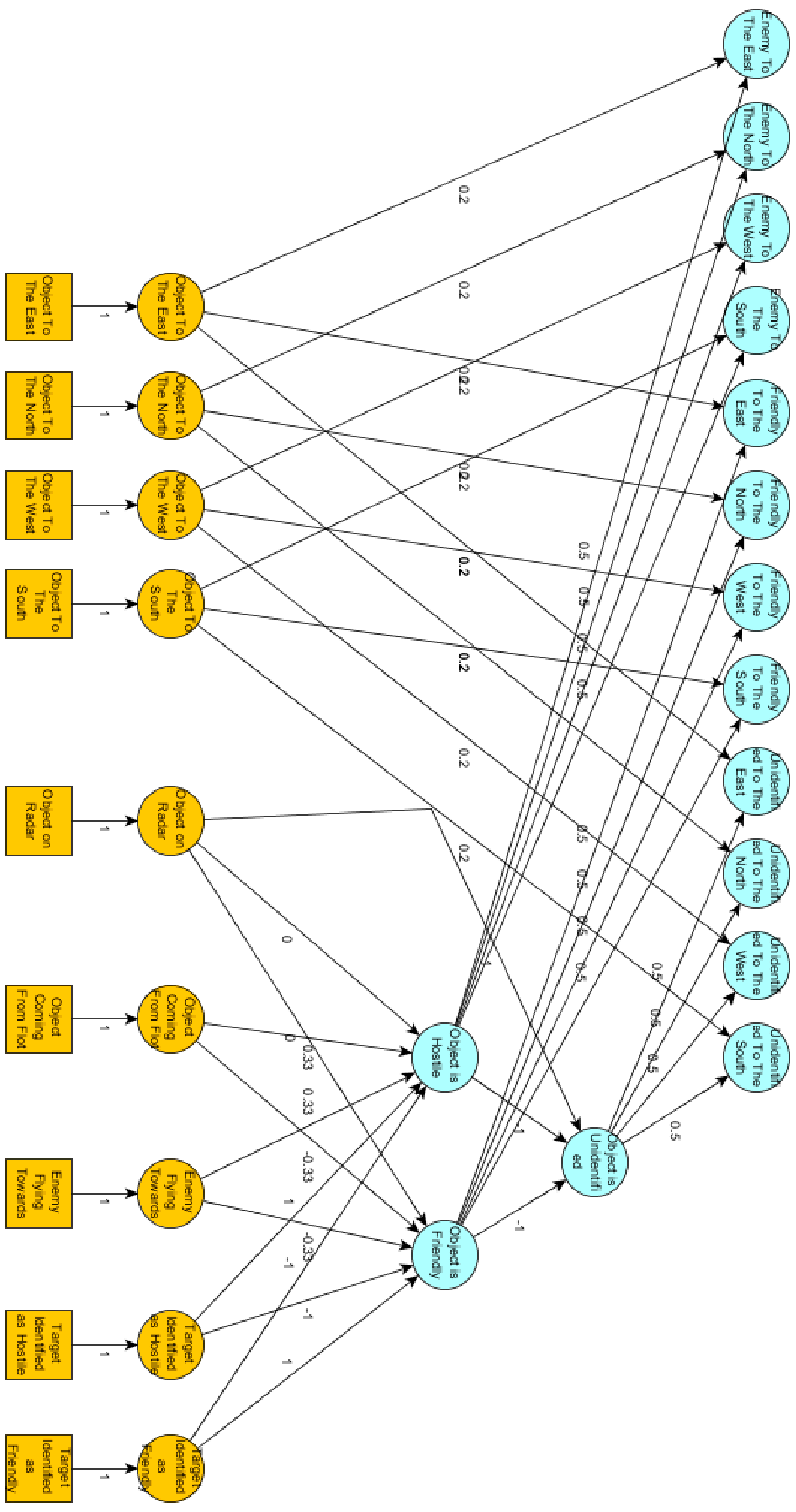


Fig. 4 Belief network for awareness scenario (weights are taken from the set {1, 0.5, 0.33, 0.2, 0}). Here, yellow squares indicate observations, yellow circles are simple beliefs and blue circles are complex beliefs.

In the awareness scenario, the belief network consists of nodes that store the location of a nearby aircraft and whether this aircraft is friendly, hostile or unidentified. Essentially, the right part of the graph takes care of the identification of the target as being hostile or friendly (given observations on the direction in which the aircraft is flying, and the identification of the target on the radar) whereas the left part addresses beliefs with respect to the direction. Finally, the top part concerns a set of aggregated beliefs about the total situation. Note that the numbers indicated in this graph are those provided by domain experts and will be tailored by the learning algorithm.

The additional DM model for the intelligent agent (introduced in Section 2.2) was implemented as a simple non-adaptive FSM and essentially comprises four states: avoid, engage, identify and patrol. If the agent beliefs it is under attack it will avoid enemy fire. If the agent does not believe it is under attack, but does believe that an enemy is sufficiently near, it will engage the target. If the agent believes that there is an unidentified plane in the neighbourhood, it will use its identification logic. If the agent does not believe any of these things it will continue its patrol. The subsequent lower level actions can be categorized as follows: (1) determine the flight path, (2) set the desired speed, (3) select a radar mode and target, and (4) fire a missile. The appropriateness of these actions is evaluated at each time step, when the SA, DM and environment models are updated. Agents are able to perform multiple tasks simultaneously (such as firing while changing their flight paths and speeds) while it is prevented that ongoing actions are accidentally cancelled when new actions are selected.

4.4 Experimental Setup

The one-versus-one scenario with the cognitive agent and the scripted agent, as described in Section 4.2, is used. The scripted agent uses a small and static set of pre-defined rules. The cognitive agent patrols until it encounters the scripted agent, which it will have to identify and, in case it is hostile, destroy.

As mentioned before, in order to be able to use the incremental learning described in Section 3, scenarios have been created for each of the behaviours that the intelligent agent is required to master in order to be successful in the one-versus-one scenario. The main focus in this paper is on learning the aforementioned awareness scenario. A single evaluation run of the awareness scenario consists of multiple trials in which a cognitive agent has encounters with a number of scripted agents. Each time the agent destroys an enemy it will gain a fitness value of 100. Each time the agent destroys accidentally a friendly target it will be given a fitness value of -100. At the end of an evaluation run of the scenario the total fitness value is divided by the number of trials in the scenario. In order to empirically compare the different learning algorithms the following steps are performed:

1. Find appropriate parameter settings for the two learning algorithms, using the awareness scenario.
2. Make a comparison between the two learning approaches on the aforementioned scenario, and also compare them with a benchmark, namely random search through the space of connection strengths. Test both approaches with the same number of iterations of the algorithm.
3. Evaluate how many different extensions in isolation improve the performance of the learning algorithms using the awareness scenario, except for the incremental learning which is performed on a number of scenarios.
4. Evaluate the performance of the best algorithm on the complete one-versus-one scenario of which the awareness scenario is a component.

The rationale for the approach presented above is to use the *awareness* scenario to get an indication of how valuable extensions in principal are, thereby assuming that the extensions work in an independent way. Studying all possible combinations would involve a huge number of experiments and would provide only a limited amount of additional information as the most promising combinations are tested together in a more extensive scenario.

Specifically, for the extensions, the following scenario-specific parts have been used: For extension 2, the scenario-specific fitness function extension essentially provides rewards for specific milestones that have been reached. In this case, the fitness function also rewards identification of the target (in addition to a reward for the destruction of it). In extension 3, symmetry information has been added for the belief network by indicating a symmetry between the different directions (North-South, etcetera) present in the network. For extension 4 (limiting the value range in the search space), only a limited number of values are allowed for the connection strengths, instead of allowing continuous values. In this case the values -1, 0, 0.2, 0.33, 0.5 and 1 are used. These values represent a negative influence, no influence and four different levels of positive influence, respectively. For extension 5 (incremental learning), learning is first limited to a scenario where the main goal is to classify the other aircraft as hostile or friendly whereas in the second part the awareness scenario should be learned. Note that extension 1 does not rely on scenario specific parts. The results of the various setups are described in Section 6.

5 Implementation

In order to be able to run the aforementioned simulations an Off-Line Learning Environment (OLLE) was used. The OLLE was created for experiments with machine learning in the air combat domain. The machine learning techniques we use, typically take a large amount of iterations to produce practical results. Therefore, it is necessary to be able to run simulations at a high speed. The programmer can define and add platforms, weapons and objects to the simulation and control the entities and the camera viewpoint. It has graphical options to show the learning progress, to view how the fitness of the agent develops, etc. The OLLE consists of four distinct parts: (1) a simulation environment, based on semi-Newtonian physics, (2) an air combat environment, that extends the simulation environment with weapon (effects), (3) scripted agents, that is, aircraft or vehicles that act on the basis of a static rule base, and (4) intelligent, adaptive agents, like the aircraft controlled by a human-like agent presented in this paper.
The setup described in Section 4 is implemented in this simulation. Note that the OLLE is a light-weight research tool for machine learning, rather than a tool that is intended for operational evaluation of equipment and tactics.

6 Empirical Results

This Section presents the results for both the basic learning algorithm as well as its extensions.

6.1 Performance Basic Learning Algorithm

Table 1 Algorithm parameters

Parameter	Evolutionary Algorithm Setting	Hill Climbing Setting
Number of experiment repeats	80	80
Number of generations	1000	1000
Mutation Rate	0.5	0.5
Mutation sigma	0.3	0.3
Population size	30	-
Number replaced per generation	1	-
Tournament size	5	-

The parameters used for the algorithms are shown in Table 1 and figure 6 shows the performance on the awareness scenario of the three different algorithms without using any extensions. Each line in the figure represents the average best fitness value in the population over 80 experimental runs. The results indicate that the evolutionary algorithm performed best (confirming our hypothesis), better than random search through the space. Hill climbing was unable to surpass the benchmark, mainly due to the fact that it has difficulty with escaping local optima.

Fig.6 Comparison of three algorithms using the awareness scenario

6.2 Performance Extensions

Table 2 shows the performance of the evolutionary algorithm with the extensions proposed in Section 3. Note that with exception of the last experiment, the extensions have been tested separately to judge their individual contributions. In the last experiment it is demonstrated that various extensions can work together to produce good results in a complex scenario, which shows promise for using the approach for different (and more complex) scenarios as well. Table 2 shows that adding extensions such as a scenario specific fitness function, limiting the search space, and incrementally learning, improved the results, whereas adding output noise and pre-defining symmetry did not. So our hypothesis is not completely supported (see also the significance results in the following subsection). The poor results for adding output noise likely stem from the fact that the number of repetitions of each scenario was kept relatively low, since the computational effort for calculating through a scenario is quite high, rendering a larger number of repetitions unfeasible. The lack of results produced by defining symmetry is most likely blamed on the specific scenario used, which prevented it to fully come to fruition.

Table 2 Experimental results using different extensions for the evolutionary algorithm

Exp. #	Scenario	Extensions	Average Best Fitness
1	Awareness	No extensions	**31.6**
2	Awareness	Adding output noise	**21.7**
3	Awareness	Scenario specific fitness function	42.6
4	Awareness	Pre-defined symmetry	**29.3**
5	Awareness	Limited search space	42.6
6	Awareness	Incremental learning	50.0
7	Awareness	Incremental learning (equal time)	**37.5**
8	Complete 1 v 1	No extensions	**28.0**
9	Complete 1 v 1	Pre-defined symmetry / Limited search space / Incremental learning	**50.0**

The most promising extensions will be discussed in a bit more detail. The improvement in experiment 3 (Table 2), using the scenario-specific fitness function is shown to be significant ($p < 0.001$ using a two-sided t-test). Limiting the search space (scenario 5) also turns out to be effective. The results show a significant improvement ($p < 0.00001$). When considering incremental learning (experiment 6 and 7) a large improvement was observed. Figure 7 shows the different approaches for incremental learning compared to each other. The difference between fully learned and equal time (experiments 6 and 7 respectively) is that in the former the agent is allowed to learn until it fully masters the first task, before starting with the second task. In experiment 7 the agent splits the 1000 evaluations between both scenarios equally, learning each scenario for 500 evaluations. It can be observed that limiting the learning time per scenario with incremental learning reduces performance. However, when spending sufficient time to learn the sub-scenarios well enough, a significant improvement can be made, when compared to the basic learning algorithms (p <0.00001).

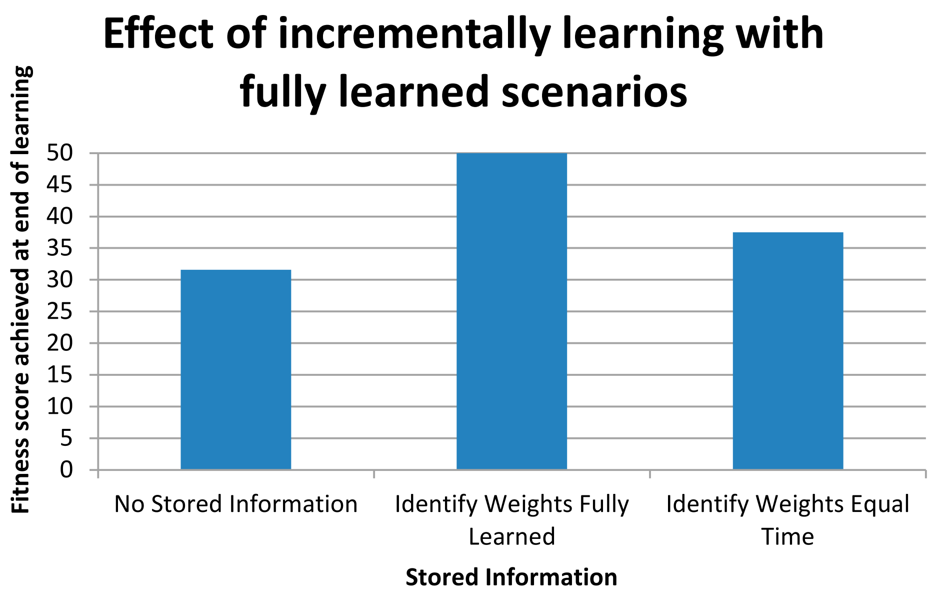


Fig. 7 Results of the different approaches for incremental learning in the Awareness Scenario

Experiments 8 and 9 (using the full scenario described in Section 4.2) in Table 2 show a comparison between the standard evolutionary algorithm and the algorithm extended with incremental learning, symmetry and limiting the values in the search space. As Table 2 shows, the extended evolutionary algorithm largely outperformed the algorithm without the extensions in the learning of the complete one-versus-one scenario. Only a limited number of runs for this scenario have been performed, because the computational resources were restricted and the algorithm was generally computationally expensive. Because of the limited number of runs and resulting small sample sizes, no significance tests could be performed.

7 Discussion

In this paper, an evolutionary learning technique has been presented with which it is possible to learn aspects of a cognitive model, thereby working under the assumption that such a cognitive model (due to its basis in cognitive science) will result in human-like behaviour. The determination of these aspects, i.e. connection strengths between beliefs in the model would previously require a substantial knowledge elicitation effort with domain experts, but these connection strengths can now be learned effectively from a simulated environment. The evolutionary technique is based upon scenarios which are configured to learn appropriate (domain dependent) connection strengths in the cognitive model. In order to test the approach, a case study has been performed in the domain of fighter pilot air combat. The results show that it is possible to apply the presented learning technique to find appropriate belief networks for cognitive models of intelligent agents (opponent fighters) in the aforementioned domain. The learning results in the required 'human-like' behaviour. In fact, the technique combines the best of two worlds: it results, to a certain degree, in human-like behaviour with a decreased need to consult domain experts. In order to further improve the learning behaviour, several extensions have been proposed, of which some have shown to be successful in the current application. The work presented here, builds forth on the work reported in [5], but provides significant augmentations. In particular, the approach presented here requires only appropriate scenarios whereas the technique presented in [5] requires domain experts to manually dictate the precise outcomes of the model being learned.

Learning approaches to obtain a sufficient level of situation awareness have been proposed before. In [9] for example, a genetic algorithm is used to obtain situation awareness. Alternative machine learning methods have been used for this purpose as well, such as particle filters that have been used for state estimation for Mars Rovers [10] and Adaptive Resonance Theory (ART) that has been used for information fusion [11]. A co-evolutionary approach has also been proposed by Smith *et al.* [16]. The latter authors use a learning classifier system, which replaces a human test pilot, with the aim of finding new flight manoeuvres for a novel combat aircraft. None of these approaches however uses cognitive models or strives towards the development of human-like behaviour.

Future work could attempt to further decrease the amount of expert knowledge necessary through additionally extending or simplifying the topology of such a cognitive model, rather than just adapting the connection strength between nodes, i.e. structure learning. It would also be of interest to compare the relative performance of the different techniques and the initial level of domain-specific expertise that is needed to eventually create expert agents. Furthermore, we want to explore how the results obtained can be transferred to other domains. Finally, we would like to investigate the learning for more complex scenarios.

8 References

1. Abt, C. (1970). Serious Games. New York: The Viking Press.

2. Hoogendoorn, M., Lambalgen, R.M. van, and Treur, J., Modeling Situation Awareness in Human-Like Agents using Mental Models. In: Walsh, T. (ed.), Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, IJCAI'11, 2011, pp. 1697-1704.

3. Harbers, M., Bosch, K. v.d., and Meyer, J.J. (2009), Modeling Agent with a Theory of Mind. In: Baeza-Yates, R., Lang, J., Mitra, S., Parsons, S., and Pasi, G., (eds.), Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Agent Technology, IEEE Computer Society Press, pp. 217 – 224.

4. Hoogendoorn, M., Merk, R.J., and Treur, J. (2010).An Agent Model for Decision Making Based upon Experiences Applied in the Domain of Fighter Pilots. In: Huang, X.J., Ghorbani, A.A., Hacid, M.-S., Yamaguchi, T. (eds.), Proceedings of the 10th IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT'10. IEEE Computer Society Press, 2010, pp. 101-108.

5. Gini, M.L., Hoogendoorn, M., Lambalgen, R.M. van (2011).Learning Belief Connections in a Model for Situation Awareness, In: Kinny, D., Hsu, D. (eds.), Proceedings of the 14th International Conference on Principles and Practice of Multi-Agent Systems, PRIMA'11. Lecture Notes in Artificial Intelligence, Springer Verlag.

6. Bosse, T., Merk, R.J., and Treur, J. (2012). Modelling Temporal Aspects of Situation Awareness. In: T. Huang et al. (eds.), *Proceedings of the 19th International Conference on Neural Information Processing, ICONIP'12, Part I*. Lecture Notes in Computer Science, vol. 7663, pp. 473–483. Springer-Verlag, Berlin Heidelberg, 2012.

7. Endsley, M.R. (1995).Toward a theory of Situation Awareness in dynamic systems. *Human Factors,* 37(1), 32-64.

8. Baluja, S. (1994).Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. CMU-CS-94-163,Carnegie Mellon University.

9. Salerno, J., Hinman, M., Boulware, D., and Bello, P. (2003). Information fusion for Situational Awareness, Proc. Int'l Conf. on International Fusion Cairns , Australia.

10. Dearden, R., Willeke, T., Simmons, R., Verma, V., Hutter, F., Thrun, S. (2004). Real-time fault detection and situational awareness for rovers: report on the Mars technology program task, Proc. Aerospace Conference, vol.2, pp. 826-840, March 2004.

11. Brannon, N., Conrad, G., Draelos, T.,Seiffertt, J., Wunsch, D., and ZhangP. (2009).Coordinated machine learning and decision support for Situation Awareness *Neural Networks*, 22(3), April 2009, pp.316-325.

12. Laird J, Rosenbloom P, Newell, A. (1987). Soar: An Architecture for General Intelligence. Artificial Intelligence 33:1-64.

13. Anderson, J.R. (1984). The architecture of cognition. Harvard University Press, Cambridge.

14. Kieras, D.E., Meyer, D.E. (1997).An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. Human-computer interaction, pp 391-438.

15. Naveh, I., Sun, R., (2006) A cognitively based simulation of academic science. Computational and Mathematical Organization Theory, pp 313-337.

16. Smith, R.E., Dike B.A., Mehra, R.K. (2000). Classier Systems in Combat: Two-sided Learning of Maneuvers for Advanced Fighter Aircraft. In: Computer Methods in Applied Mechanics and Engineering 186(2-4):421-437.

17. Stanley, K., Miikkulainen, R. (2002). Evolving Neural Networks through Augmenting Topologies. Evolutionary Computation 10(2):99-127.

18. Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Publishing Co., Inc., Redwood City, CA.

19. http://www.presagis.com/products_services/products/modeling-simulation/simulation/stage/, page downloaded on November, 15, 2013.

20. Merk, R. (2010). A Computational Model on Surprise and Its Effects on Agent Behaviour in Simulated Environments., in Advances in Practical Applications of Agents and Multiagent Systems (PAAMS), 2010.

21. Jones, R.M., Laird, J.E., Nielsen, P.E., Coulter, K.J., Kenny, P., Koss, F.V. (1999). Automated Intelligent, Pilots for Combat Flight Simulation. AI Magazine ,Volume 20, Number 1, 1999 American Association for Artificial Intelligence, pp27-41.

22. Spronck, P., Ponsen, M., Sprinkhuizen-Kuyper, I., & Postma, E. (2006). Adaptive game AI with dynamic scripting. Machine Learning, 63(3), 217–248. doi:10.1007/s10994-006-6205-6

23. Toubman, A., Roessingh, J.J.M., Spronck, P.,Plaat, A., Herik, H.J. van den(2014). Dynamic Scripting with Team Coordination in Air Combat Simulation. Paper presented at the 27th IEA-AIE 2014. Proceedings of the 27th International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems, LNCS, Springer Verlag.

24. Koopmanschap, R. (2012).Improving a Model for Situation Awareness in CGF Fighters with Machine Learning Techniques, Master Thesis, VU University Amsterdam, 2012.

25. Simon, H.A. (1992). What is an “explanation” of behavior? Psychological Science, Vol.3, No. 3, 150-161.

26. Yao, X., Evolving Artificial Neural Networks, Proceedings of the IEEE, 87:9 pp. 1423-1447, 1999

Acknowledgements

Lt. Col. Roel Rijken (Royal Netherlands Air Force) created a first version of the Off-Line Learning Environment (OLLE) used in this study and provided valuable advice. The authors also thank Robbert-Jan Merk, Remco Meiland and Pieter Huibers at the National Aerospace Laboratory NLR for their feedback and assistance with the simulation environment. Finally, the authors thank two anonymous reviewers for carefully scrutinizing the study. This research has greatly benefitted from their help.

This page is intentionally left blank.

WHAT IS NLR?

The NLR is a Dutch organisation that identifies, develops and applies high-tech knowledge in the aerospace sector. The NLR's activities are socially relevant, market-orientated, and conducted not-for-profit. In this, the NLR serves to bolster the government's innovative capabilities, while also promoting the innovative and competitive capacities of its partner companies.

The NLR, renowned for its leading expertise, professional approach and independent consultancy, is staffed by client-orientated personnel who are not only highly skilled and educated, but also continuously strive to develop and improve their competencies. The NLR moreover possesses an impressive array of high quality research facilities.

***NLR** – Dedicated to innovation in aerospace*