ATLANTIS PRESS International Journal of Computational Intelligence Systems, Vol. 10 (2017) 804–814

Tactics Exploration Framework based on Genetic Programming

Jian Yao , Weiping Wang , Zhifei Li , Yonglin Lei , Qun Li *

College of Information System and Management
National University of Defense Technology
137 Yanwachi, Changsha, Hunan 410073, China
E-mail: liqun@nudt.edu.cn

Received 9 September 2016

Accepted 20 February 2017

Abstract

Engagement-level simulation is a quantitative way to evaluate the effectiveness of weapon systems before construction and acquisition, minimizing the risk of investment. Though contractors have built simulation systems with high fidelity models of weapon systems and battlefields, developing competent tactics to give full play to new weapon systems in simulation experiments is labor intensive, as most classical tactics tend to be out of date. In this work, we proposed a tactics exploration framework (TEF) that applied grammar-based genetic programming (GP) to generating and evolving tactics in the engagement-level simulation. Tactics are represented with modular behavior trees (BTs) for compatibility with the genetic operators. Experiments to explore submarine tactics have been conducted to observe and study the exploration process. The experimental results show that the TEF based on GP is efficient to explore tactics in the formalism of BTs.

Keywords: tactics exploration framework; grammar-based genetic programming; behavior trees; submarine warfare simulation

1. Introduction

The construction of a new conceptual weapon system requires a large budget and an extended period. Engagement-level simulation is a quantitative method to evaluate the effectiveness of a weapon system before its construction and acquisition, minimizing the investment risk and also providing feedback to optimize the technology solutions[1].

Before finalizing the design, various technology solutions for new weapon systems need to be evaluated in multiple engagement scenarios. Classical tactics are no longer applicable to operate weapon systems utilizing advanced technologies, and test scenarios show that no universal tactic performs well under these conditions; consequently, a lot of competent tactics are required to conduct a comprehensive evaluation. However, developing tactics requires significant domain expertise, and scripting tactics is time-consuming and labor-intensive in practice[2]. Therefore subject matter experts (SMEs) need a more efficient method to develop competent tactics.

Evolutionary algorithms (EAs)[3] are a class of population-based stochastic search techniques to search complex problem for optimal solutions, providing a potential framework to explore engagement-level tactic space. However, tactics are more about structures (e.g., the goal hierarchy and action sequence) than numerical parameters, and

* Corresponding author.
Email: liqun@nudt.edu.cn.

classical EAs focus on numerical optimization and lack the ability to explore structure space.

Genetic programming (GP)[4,5], as an extension of the classical genetic algorithm (GA)[6], is defined as a structural optimization algorithm[7], which evolves solutions represented with executable programs instead of linear strings. Grammar-based GP[8] redefined the elements on a derivation tree with a problem-oriented context-free grammar (CFG) instead of Lisp expressions as in the original GP. CFG provides a general representation structure for defining constraints and limits in problem solutions.

The advantages of grammar-based GP include the following: 1) formal grammar restricts the search space; 2) problem-oriented grammar provides a flexible representation for domain knowledge and problem solutions; and 3) grammar-based GP extends GA with the ability of structure optimization. In summary, grammar-based GP is a promising framework to explore tactic space, as tactics are both domain knowledge intensive and representation structured.

In this paper, we proposed our tactics exploration framework (TEF) based on grammar-based GP to evolve tactics in simulation, in which a grammar of behavior tree (BT)[9] formalism is used for tactic representation. BTs are a formal method to represent both goal hierarchies and action sequences in tactics; moreover, the modular tree structure is inherently compatible with genetic operators in grammar-based GP. To validate our method, an undersea warfare scenario is built to explore submarine tactics in the engagement-level simulation.

The outline of this paper is as follows: Section 2 presents some related work. Section 3 is an overview of our proposed framework to explore submarine tactics with grammar-based GP. Section 4 introduces BTs and details their specific application to submarine warfare. Section 5 presents a grammar-based GP technique to evolve tactics, followed in Section 6 by submarine tactics exploration experiments to validate our method and also the analysis of experimental results. Conclusions and the future works are presented in Section 7.

2. Related Work

In most military simulation systems, tactics are script-based. SMEs propose new tactics and represent them with informal paper-based diagrams before coding them into scripts to do the simulation. The most often-used tactics representations are rule-set, decision tree, and finite state machine (FSM). These are criticized for their defects in practice[10,11,12], such as the inability to scale up in the complex domain, the difficulties for SMEs to understand and reuse tactic scripts, and the tedious scripting.

To address these problems, software engineering is introduced to provide domain specific language (DSL)[13] for tactics representation with automatic script generation. For example, Kerbusch implemented a simple FSM with the scripting language Lua to describe air combat tactics[14]. Evertsz proposed a Tactics Development Framework (TDF)[2,12], which fully defined the tactical elements, such as missions, actors, roles, and scenarios. In TDF methodology, a goal structure is designed to decompose the mission objectives into sub-goals, and a plan diagram based upon activity diagrams is provided to describe the procedure of tactic. However, it is noteworthy that DSL is just a method to represent tactics, and the tactics development still relies heavily on domain expertise.

Developing tactics for a new weapon system used to be an "trial and error" process; that is costly and time-consuming for SMEs to propose, test, and redesign tactics. Therefore, technologies in artificial intelligence (AI), such as EA and reinforcement learning (RL), are introduced to optimize and learn tactics in the simulation.

EA, as a general search technique, is widely used to optimize tactics in the military domain. Mulgund applied GA to optimize the team formation and intercept geometry in large-scale air combat[15]. Liang also used GA to design anti-torpedo tactics[16]. Similar works can be found in[17,18,19]. In these applications, SMEs built the structures of tactics and implemented EAs to optimize the critical parameters on the structures; however, EAs did little exploration in tactic structures.

While EA searches for optimal tactics through

evolving a population of individual agents, RL enables an agent to improve its tactic through training in the simulation. Teng proposed the FALCON[20] architecture based on the self-organizing neural network to train strategies in 1-v-1 dogfight training simulation. The proposed models showed significantly improved adaptivity and higher performance in human-in-the-loop (HIL) experiments. Toubman applied dynamic scripting (DS)[21] to improving team coordination behavior in air combat and got a higher performance in experiments[22]. DS selects high weight rules from a rule base to build scripts for combat simulation; then the results are fed back to update the weight of rules. This process continues until a reliable script is reached. However, tactics represented with weighted rule-set models or neural network models were black box systems, from which it is difficult for SMEs to understand the internal logic and get explicit tactics[23].

3. Tactics Exploration Framework

This section presents an overview of our proposed framework to explore tactics, including components and workflow. TEF consists of an engagement-level simulator to conduct experiments and a grammar-based GP as a tactic exploration engine. A tactics representation tool (TRT) is also provided for SMEs to edit and read tactics represented with BTs diagrams.

TEF is based on an engagement-level simulator, named Weapon Effectiveness Simulation System (WESS)[24], which supports submarine warfare with high-fidelity models of the submarine, sonar, torpedo, etc. Tactics are defined with Python for execution, a scripting language that is close to a natural language. The Python interfaces for state query and action execution are listed out as the basic elements in tactic scripts. State variables in queries are extracted from onboard instruments, and available actions to execute are basic orders from control systems as in real-life combat.

Tactic representation is a core issue in a combat simulation. In TEF, we use the BTs to represent tactics. Compared to other forms of representation (e.g., rule-set, FSM), BTs enable SMEs to represent both the goal hierarchies and action sequences of a tactic in one diagram. The inherent advantage comes from the modularity of the tree structure in that the subtrees can be composed into new BTs without limits. Moreover, both BTs and the derivation trees of GP are in the tree structure. Thus the genetic operators of GP are compatible with BTs in semantics. A TRT is also developed as the bridge between SMEs and GP, which enables SMEs to edit domain knowledge for GP and also read the tactics generated from the evolution of GP.

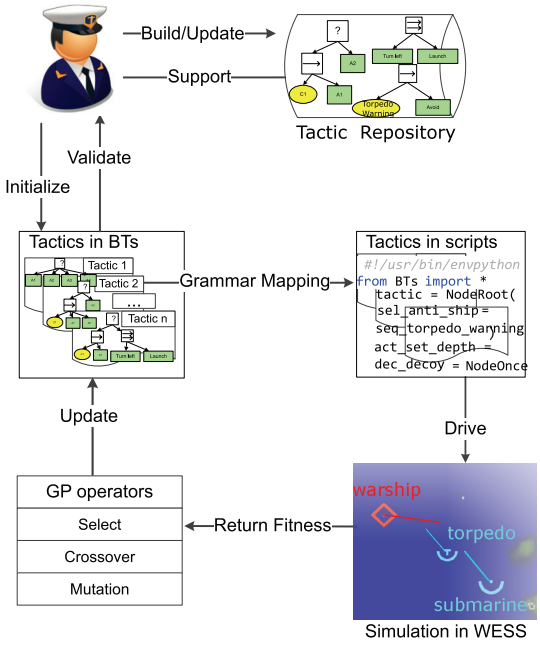


Fig. 1. Tactics Exploration Framework

Grammar-based GP works as the engine in TEF, which produces tactics to conduct simulations and updates tactics with feedback engagement results. To utilize grammar-based GP in tactic exploration, a context-free grammar of BT formalism in Backus-Naur form (BNF) is defined to represent tactics, and the genetic operators are improved to stabilize the evolution process. The details are presented in Section 5.

The workflow of TEF in Fig. 1 proceeds as follows. First, a repository of the most used tactics and tactic modules in doctrine is built up. Next, SMEs draft initial generation of tactics in BT diagrams based on the tactic repository. The diagrams are mapped to derivation trees for genetic manipula-

tion and translated into executable tactic scripts with TRT. After that, simulations are conducted to evaluate the fitness of tactics in combat. Guided with the feedback fitness, GP produces a new generation of tactics with genetic operators and starts new simulations to iterate the exploration. SMEs validate new tactics in exploration to update the tactic repository.

4. Tactic Representation

4.1. Tactic Definition

The tactic definition varies in different warfare levels, and there is no unified definition.

At the campaign level, Clausewitz defined tactics as "*the science and art of organizing a military force, and the techniques for combining and using weapons and military units to engage and defeat an enemy in battle*[25] ". In the Department of Defense (DoD) Dictionary of Military and Associate Terms, tactics are defined as "*the employment and ordered arrangement of forces in relation to each other. Also procedures, techniques* ".

At the engagement level, Shaw stated that air combat tactics are about "*choice of attack formations, pre-attack positioning, attack speed, maneuvering to attain a firing position, and engagement/disengagement decision criteria*[26] ".

For this paper, we simplified the engagement-level definition that tactics are a sequence of actions to achieve a specific goal.

4.2. Behavior Tree Formalism

BTs originated from the computer game industry as a more modular, scalable, and reusable alternative to FSM in the development of AI components[9]. Recently, the robotics community has shown great interest in BTs as a modular control formalism for unmanned aerial vehicles (UAVs) and complex robots[27]. Colledanchise *et al.* provided an accurate mathematical formula for BTs, and further analyzed the safety, efficiency, and robustness of composite BTs[28].

A BT is a directed rooted tree defined as a tuple $BT = \langle V, E \rangle$ where

$V = A \cup C \cup N \cup \tau$ is the finite set of nodes with action nodes A, condition nodes C, control flow nodes N, and a root node τ.

$E \subset V \times V$ is a finite set of edges, $\forall \langle v_i, v_j \rangle \in E$, $v_i, v_j \in V$, v_i is the parent node of v_j, and v_j is the child of v_i.

The algorithms of BT nodes are presented in Algorithm $1-7$. The symbols are $S(Success), F(Failure), R(Running) \subseteq X$, state space $X(t) \in X$, and control signals $U(t) \in U$.

The route from the top level to each leaf represents one course of action, and the behavior tree algorithm searches among those courses of action in a left-to-right manner. In other words, it performs a depth-first search.

The execution of a BT proceeds as follows. The root node sends signals called "ticks" to its children at a certain frequency. This tick is then passed down to leaf nodes (***Action*** or ***Condition***) with the guide of control flow nodes. Once a leaf node receives a tick and executes its task, it returns to its parent a status *Running* if its execution has not finished yet, *Success* if it has achieved its goal, or *Failure* otherwise, and a ***Condition*** queries about the state, and an ***Action*** performs a specific task in the execution.

Control flow nodes are typically ***Selector***, ***Sequence***, ***Parallel***, and ***Decorator***; the first executes all its children from left to right until one fails, while the second executes its children from left to right until one succeeds. A ***Parallel*** node executes all its children sequentially, returns *Success* if a given number of children return *Success*, returns *Failure* if the remaining running children are not enough to reach the given number (even if they are all going to succeed), and returns *Running* otherwise. The ***Decorator*** sets constrains to pass ticks to its children.

Algorithm 1: Selector

```
for i ← 1 to N do
    state ← Tick(child(i));
    if state == Running then
        return Running;
    end
    if state == Success then
        return Success;
    end
end
return Failure;
```

Algorithm 2: Sequence

```
for i ← 1 to N do
    state ← Tick(child(i));
    if state == Running then
        return Running;
    end
    if state == Failure then
        return Failure;
    end
end
return Success;
```

Algorithm 3: Parallel

for $i \leftarrow 1$ *to* N **do**
  $state_i \leftarrow Tick(child(i))$;
end
if $nSuccNodes(state_i) \geqslant nS$ **then**
  return *Success*;
else if $nFailNodes(state_i) \geqslant nF$ **then**
  return *Failure*;
else
  return *Running*;
end

Algorithm 4: Action

if $X_n(t) \in S_n$ **then**
  return *Success*;
end
if $X_n(t) \in F_n$ **then**
  return *Failure*;
end
if $X_n(t) \in R_n$ **then**
  $U_n(t) \leftarrow \gamma_n(X_n(t))$;
  return *Running*;
end

Algorithm 5: Condition

if $X_n(t) \in S_n$ **then**
  return *Success*;
end
if $X_n(t) \in F_n$ **then**
  return *Failure*;
end

Algorithm 6: Decorator

if $Check(constrains) ==$ ***True*** **then**
  $state \leftarrow Tick(child)$;
  return *state*;
else
  return *Failure*;
end

Algorithm 7: Root

return $Tick(child(0))$;

4.3. *Representing Submarine Tactics with Behavior Trees*

According to the definitions, tactics are goal-oriented and procedural in nature. Thus the representation focuses on the decomposition of goals and the sequence of action in the procedure.

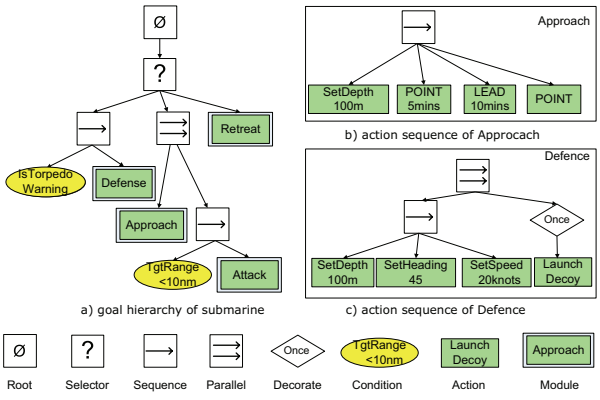


Fig. 2. An example of submarine tactic in Behavior Tree formalism

4.3.1. *Goal Hierarchies*

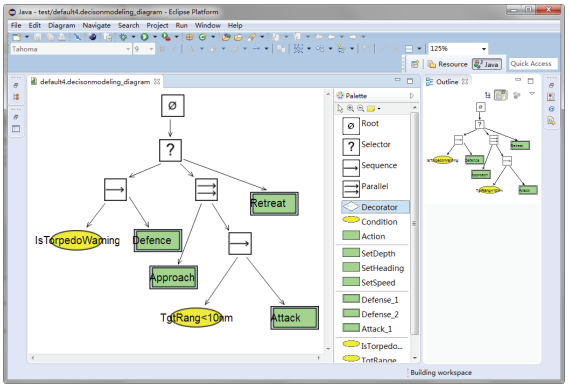
BTs provide tree structure to support top-down decomposition from mission goal to subtasks, state queries and actions. In turn, low-level subtasks can also be composed into higher-level tasks to achieve specific goals through bottom-up integration.

In littoral defense, the tactical goal of the conventional submarine is to destroy or clear out warships in the operation area. In general, submarine tactics can be decomposed into approach, attack, defense, and retreat, as shown in Fig. 2(a). For example, the approaching task is further decomposed into several legs to locate and chase a warship, and each leg is composed of actions to set heading and speed. Classical tactic modules in the repository, such as an effective torpedo defense, can be added to a new tactic or replace the same part in an old one.

4.3.2. *Action Sequences*

The sequence of actions is defined with control nodes in BTs; *Sequence* and *Selector* for serial actions, *Parallel* for concurrent actions, and *Decorator* for specific temporal actions.

In submarine warfare, a tactic consists of maneuver and fire control. *Sequence* executes subtasks or actions with a strict time order, such as legs to locate the warship[1]. Fig. 2(b) is a simple Point-Lead-Point leg. *Selector* defines the priorities of subtasks or actions; for example, submarine interrupts attacking maneuver to high prior torpedo defense, shown in Fig. 2(b). Fire control, including sensor management, weapon launch, and countermeasure release, is concurrent with the maneuver. As in the torpedo defense, submarine maneuvers to avoid torpedo and releases decoys at the same time, as shown in Fig. 2(c) with a *Parallel*. Moreover, actions with specific constraints on intervals and times are defined with *Decorator*.



a) Screeshot of TRT Eclipse plugin

```
from bt import *  #import behaviour tree package
from Common import * #import simulation Interfaces
def InitTactic(Platform):
    #Build the Tactic
    tactic = NodeRoot(root)
    sel_anti_ship  = NodeSelector(root)
    seq_torpedo_warning = NodeSequence(sel_anti_ship)
    cond_is_torpedo_warning = NodeCondition(seq_torpedo_warning,IsTorpedoWarning,equal,True)
    para_defence  = NodeParallel(seq_torpedo_warning)
    seq_defence_maneuver  = NodeSequence(para_defence)
    act_set_depth_100 = NodeAction(seq_defence_maneuver,SetDepth,100)
    act_set_heading_45 = NodeAction(seq_defence_maneuver,SetHeading,45)
    act_set_speed_20 = NodeAction(seq_defence_maneuver,SetSpeed,20)
    dec_decoy = NodeOnceDecorator(para_defence)
    act_decoy = NodeAction(dec_decoy,LaunchDecoy)
    para_approach_attack = NodeParallel(sel_anti_ship)
    seq_approach = NodeSequence(para_approach_attack)
    act_setdepth_100 = NodeAction(seq_approach,SetDepth,100)
    act_POINT_5 = NodeAction(seq_approach,POINT,5)
    act_LEAD_10 = NodeAction(seq_approach,LEAD,10)
    act_POINT = NodeAction(seq_approach,POINT)
    seq_attack = NodeSequence(para_approach_attack)
    cond_tgtRange_less_10 = NodeCondition(seq_attack,TgtRange,less,10)
    seq_attack_launch = NodeSequence(seq_attack)
    act_set_depth_50 = NodeAction(seq_attack_launch,SetDepth,50)
    dec_torpedo = NodeIntervalDecorator(seq_attack_launch,3)
    act_torpedo = NodeAction(dec_torpedo,LaunchTorpedo)
    para_retreat = NodeParallel(sel_anti_ship)
    dec_decoy = NodeOnceDecorator(para_retreat)
    act_decoy  = NodeAction(dec_decoy,LaunchDecoy)
    seq_retreat = NodeSequence(para_retreat)
    act_set_heading_45 = NodeAction(seq_retreat,SetHeading,45)
    act_set_depth_100 = NodeAction(seq_retreat,SetDepth,100)
    act_set_speed_10 = NodeAction(seq_retreat,SetSpeed,10)
    Platform.SetTactic(tactic)
def RunTactic(Platform):
    #Run the Tactic
    Platform.GetTactic().Run()
```

b) Tactic script generated by TRT

Fig. 3. Tactics representation tool and generated tactic script

4.4. *Tactics Representation Tool*

Based on BT formalism, we developed a DSL for tactic representation with Eclipse Modeling Framework (EMF)[29]. First, the abstract syntax of DSL was developed with Ecore metamodeling, which defined both the domain-independent elements of BTs formalism and the domain elements, such as actions and state variables. Following this, a graphical editor with concrete syntax was built with Graphical Modeling Framework (GMF)[30], which provided SMEs a graphical user interface to design tactic models. Finally, a code generator was implemented with Acceleo[31], which mapped tactic diagrams into Python scripts. TRT was an Eclipse-based plugin based on the components above. See Ref. 32 for more details.

TRT provided features such as friendly tactic editor environment and flexible domain knowledge management. Tactic diagrams also promoted user comprehension and the potential for tactics reuse and sharing. Fig. 3 showed the snapshot of TRT and an example of generated script.

In TEF, TRT bridged the GP algorithm and SMEs by automatically translating between tactic diagrams and scripts. Domain knowledge, such as classical tactic modules, represented by diagrams were translated into scripts for GP manipulation. On the other hand, tactic scripts generated from GP evolution were translated back to diagrams for expert analysis and validation.

5. Exploring Tactics with Grammar-based Genetic Programming

5.1. *Grammar-based Genetic Programming*

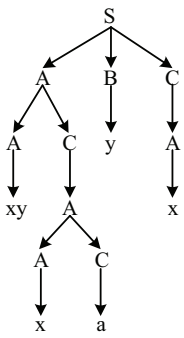
In this work, we implemented CFG-GP[33,34] to explore tactics, in which a CFG of BT formalism was defined to generate the derivation trees, and extended genetic operators are proposed to stabilize the evolution.

<S>::= <A>|<A><B><C>
<A>::=x|xy|<A><C>
<B>::=y|z|<C>
<C>::=a|b|c|<A>

(a) the production of grammar

<S> →<A><B><C>
→<A><C>y<A>
→xy<A>yx
→xy<A><C>yx
→xyxayz

(b) derivation steps



(c)derivation tree

Fig. 4. An application of grammar to generate a derivation tree

In the terminology of grammar-based GP, the genotype is a derivation tree in the language defined by a problem-oriented grammar, while the

phenotype refers to a program that produces the behavior of an individual. The grammar defines the interpretation between genotype and phenotype. Fig. 4 showed an example to generate a derivation tree from a grammar. In the evolution process, genetic operators manipulate genotypic derivation trees, while the fitness of individuals is calculated based on the performance of phenotypic programs in combat simulations.

As for any EAs, an initial population of individuals is created at random, and each individual is executed to ascertain its fitness. The individuals with higher fitness get more possibilities of being selected as parents to generate a new population through selection, crossover, and mutation operators. The evolution loop is run until a certain termination condition is met (e.g., obtaining an acceptable solution, or a maximum number of generations is reached).

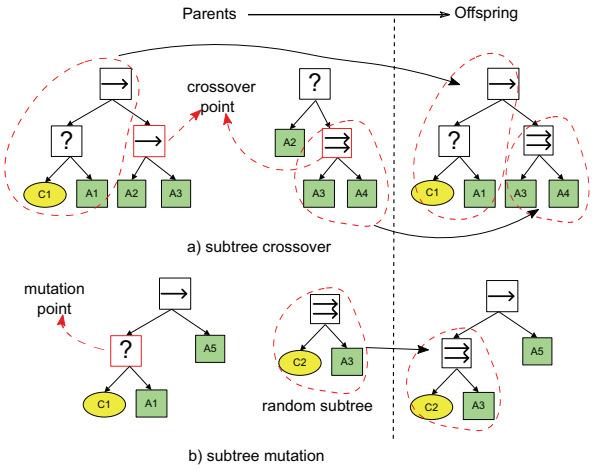


Fig. 5. Examples of subtree crossover and subtree mutation

5.2. *Grammar for Submarine Tactics*

Every problem has its own grammar, which represents the program syntax. The CFG for submarine tactics of BT formalism can be represented in BNF. $Sub_{BNF} = \langle T, G, S, P \rangle$, where

$T = A \cup C$ is the terminal set, include ***Condition*** C and ***Action*** A;

$G = N \cup \tau$ is the set of non-terminals, include ***Control*** nodes N and ***Root*** node τ;

$S = \tau$ is the start symbol (the ***Root*** node in BTs);

P is a set of production rules.

The production rules in BNF for submarine tactics are defined as follows.

$\langle Root \rangle ::= \langle BT \rangle$
$\langle BT \rangle ::= \langle BT \rangle \langle Node \rangle | \langle Node \rangle$
$\langle Node \rangle ::= \langle BT \rangle | \langle Control \rangle | \langle Condition \rangle | \langle Action \rangle$
$\langle Control \rangle ::= Selector | Sequence | Parallel | \langle Decorator \rangle$
$\langle Decorator \rangle ::= \langle DecoratorType \rangle \langle Node \rangle$
$\langle DecoratorType \rangle ::= Once | Repeat | Interval10s | \cdots$
$\langle Condition \rangle ::= \langle State \rangle | \langle StateVariable \rangle \langle op \rangle (number)$
$\langle State \rangle ::= TorpedoWarning | TorpedoReady |$
$LaunchAuthorized | ActivSonarOn | \cdots$
$\langle StateVariable \rangle ::= Depth | Speed | Heading | TgtType |$
$TgtHeading | TgtDepth | \cdots$
$\langle op \rangle ::== | < | > | <= | >= |$
$\langle Action \rangle ::= SetSpeed | SetHeading | SetDepth |$
$LaunchDecoy | ActivateJammer | \cdots$

5.3. *Genetic Operators*

5.3.1. *Crossover*

The genotype in CFG-GP is a derivation tree, and the typical form of crossover is a subtree crossover. Two individuals called the parents are selected from the population. A crossover node is chosen at random in each parent, and the subtrees rooted at the crossover nodes are swapped between parents to generate child individuals, as shown in Fig. 5(a).

The tree structure is more sensitive to the changes on high-level nodes than on low-level ones. In contrast, the high-level goal hierarchies and action sequences of tactic BTs are more stable than the lower ones. In fact, frequent changes on high-level nodes of tactic BTs trend to generate tactics that are syntactically correct but obviously violate the doctrine and perform stupid actions, leading to an unstable evolution.

As a consequence, we used a variable probability for crossover operator at different levels to keep the BT structure stable. For a BT with N levels, the top node (root) is level 0 and the lowest leaf node is level $N-1$. Assuming P_x is the crossover probability parameter for GP, we set $P_x(i) = P_x * (1 - e^{-i})$ as the crossover probability for nodes at level i.

Most tactic modules (e.g., $Approach$) are self-contained blocks, in which the actions and conditions are customized. For example, the submarine would not launch torpedoes in $Defense$. The crossover between different modules destroys the integrity of tactic modules.

To overcome this defect, we introduced an $\langle XO \rangle$ to

separate tactic modules and indicate the crossover boundaries. In the evolution process, the crossover is performed on the same tactic modules, in which *Approach* exchanges internal actions with *Approach* in other tactics and avoid random exchange with *Attack*.

$$\begin{aligned}\langle Tactic\rangle ::= \quad & Defence = \langle BT\rangle;\langle XO\rangle;\\ & Approach = \langle BT\rangle;\langle XO\rangle;\\ & Attack = \langle BT\rangle;\langle XO\rangle;\\ & Retreat = \langle BT\rangle;\langle XO\rangle;\end{aligned}$$

5.3.2. Mutation

A mutation operator selects a mutation node at random in a single parent, removes the subtree at that point, and inserts a new derivation tree generated with the CFG, as shown in Fig. 5(b).

Just as the same as the crossover, we used a variable probability for mutation operator to decrease changes on the high-level nodes. Assuming P_m is the mutation probability parameter, we set $P_m(i) = P_m * (1 - e^{-i})$ as the mutation probability for nodes at level i.

6. Case Study

6.1. Submarine Warfare Scenario

A littoral area submarine warfare scenario was built to validate our proposed method. In the scenario, a patrolling submarine dived underwater to ambush a warship. As an opponent, the warship cruised with active sonar working and attacked any submarines within its torpedo range.

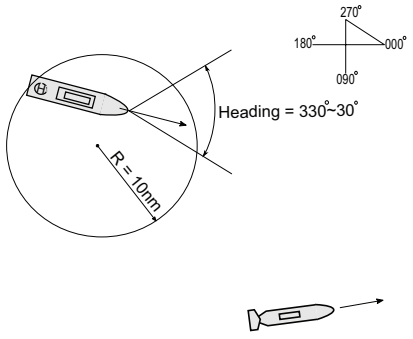


Fig. 6. Scenario of submarine warfare

To model the uncertainties in warfare, the position, speed, and heading of an adversary warship were initialized randomly in scenarios. As shown in Fig. 6, the initial position was randomly distributed in a circle with a radius of 10 nautical miles (nm). The speed was sampled from a uniform distribution on [10, 30] knots, and the heading of the warship was sampled from a uniform distribution from 330 to 30.

6.2. Experimental Setup

6.2.1. Comparison experiments

Both a submarine on active service (Sub_{Active}) and a new submarine (Sub_{New}) in design phase were tested in scenarios as comparison experiments. The baseline of the experiments was 20 expert submarine tactics based on doctrine for Sub_{Active}. The performance of grammar-based GP in tactics exploration was evaluated with a comparison to the baseline. The experiment for Sub_{New} further studied the capability of grammar-based GP to exploit the advantages of new technologies.

6.2.2. Algorithm parameters

The parameters for GP are given in Table 1. Each tactic ran 10 times in the simulation to get an average fitness value for its performance. The exploration process iterated a total of 100 generations. *Elite_size* specifies the number of individuals who received high fitness value that are guaranteed to survive to the next generation. Tournament selection chooses each parent by choosing *Tournament_size* players at random and then choosing the best individual out of that set to be a parent. *Maximum tree height* limits the height of derivation trees in the GP evolution.

Table 1. Parameters for genetic programing.

Parameter	Value
Population_size	100
Generations	100
Elite_size	15
Tournament_size	4
Single point crossover ratio	80%
Two point crossover ratio	20%
Mutation probability	5%
Maximum tree height	15

6.2.3. Fitness function

The fitness function to evaluate tactics was defined with the final result of engagement. A tactic got 1 point for 1 : 0 when the submarine eliminated the enemy without

loss and 0 for a loss without eliminating the enemy. As for a draw, we set $1/3$ for $0:0$ that both sides survived, and a higher $2/3$ for $1:1$ to explore aggressive tactics and avoid evolution in negative escape tactics.

6.2.4. *Initialization*

It was labor intensive for SMEs to initialize 100 tactics. To fully use the domain expertise and reduce the computation, the 20 expert tactics were used as seeds to generate another 80 tactics with crossover and mutation using the same parameters in Table 1. Therefore we received a total population of 100 tactics to start the evolution.

6.3. *Results*

Because of the sensitive nature of the simulations, the results analysis focused on the efficiency of tactics in the exploration process rather than the specific tactics. Fig. 7. visualizes the experimental results with the average fitness of the tactics in the exploration.

6.3.1. *Baseline*

The 20 expert tactics were tested in simulation to get the baseline, with 0.513 for Sub_{Active} and 0.636 for Sub_{New}. It means that new technologies in Sub_{New} brought about 23.9% increment in capability even run with expert tactics for Sub_{Active}.

6.3.2. *Exploration results*

In Fig. 7, it was obvious that the exploration began with a low average fitness value, the reason was that 80% of initial tactics were generated randomly. However, the average fitness value shows an uptrend in the following generations. In the Sub_{Active} experiment, the tactics started with 0.285 and rose to about 0.451 after 50 generations and then stagnated across the subsequent 50 generations. It was almost the same in the Sub_{New} experiment, with tactics raised from 0.245 to 0.55 before 70 generations. The evolution processes for both Sub_{Active} and Sub_{New} show an obvious uptrend on average fitness with decreasing standard deviation. It demonstrates the capability of grammar-based GP to explore tactics with a stable process.

In our experiments, the average fitness of both Sub_{Active} and Sub_{New} did not make evident breakthroughs over the baseline. However, the top 15% tactics show some differences. The top 15% tactics performance and expert tactics in the Sub_{Active} experiment mean that the expert tactics gave full play to Sub_{Active}, and GP did not make any breakthrough. In contrast, the top 15% tactics for Sub_{New} show slightly higher efficiency than expert ones in the last 30 generations. A one-sided *t-test* rejected the null hypothesis that the top 15% with average fitness equaled baseline (0.636) at the 5% significance level and accepted the alternative hypothesis of the top 15% with an average fitness greater than the baseline. From this observation, it can be concluded that expert tactics worked well but did not make full use of Sub_{new}; however, GP found new tactics to exploit the capability of Sub_{New} further.

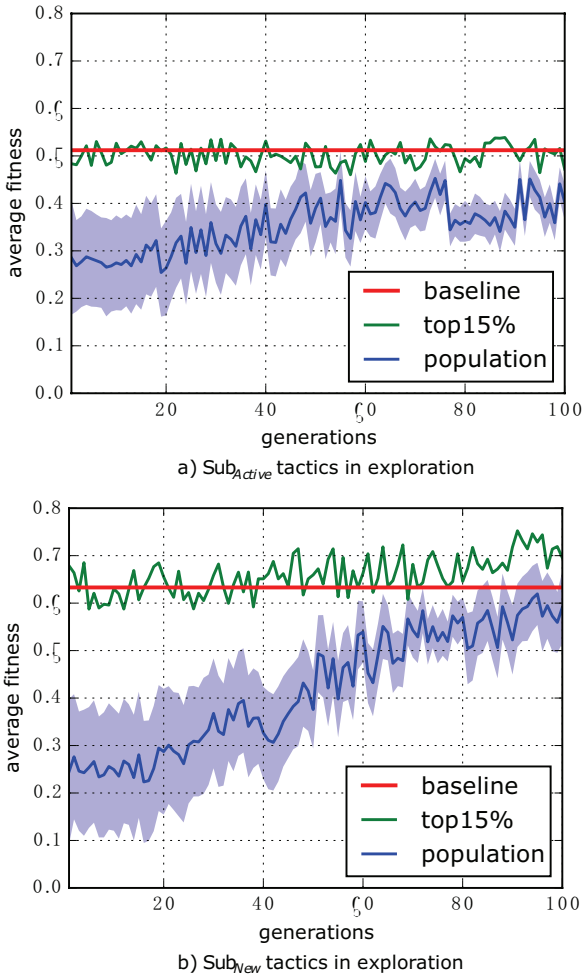


Fig. 7. Fitness of tactics in the exploration

7. Conclusions and Future Work

In this paper, we have presented a framework of grammar-based GP to explore tactics in the engagement-level simulation. BT formalism for tactics representation are compatible with GP manipulation in exploration, and also provide high-level diagrammatic goal hierarchies and action sequences. Extensions on genetic operators were proposed to stabilize the evolution process. Experiments in submarine warfare demonstrated the capability of TEF to explore tactics with incremental improvement on engagement results.

The directions for future work will focus on the efficiency of grammar-based GP, the validation of generated tactics, and the adaptiveness of tactics in uncertain combat. First, various enhancement techniques on grammar-based GP, such as extended genetic operators and grammar design, will be tested and compared in this project. The effect of GP parameters will also be analyzed with more experiments. Then, SMEs will qualitatively evaluate the feasibility of the generated tactics in real combat. Mining new tactic modules from the generated tactic trees will also be an interesting work. Last, RL techniques will also be introduced to improve the adaptivity and flexibility of tactics to make actions in uncertain combat.

Acknowledgments

This work is partly supported by the National Natural Science Foundation of China (no. 61273198 and no. 71373282). The authors acknowledge Qiwang Huang for his detailed and helpful advices to the paper.

References

1. Myeongjo Son and Taewan Kim. Maneuvering control simulation of underwater vehicle based on combined discrete-event and discrete-time modeling. *Expert Systems With Applications*, 39(17):12992–13008, 2012.
2. Rick Evertsz, John Thangarajah, Nitin Yadav, and Thanh Ly. A framework for modelling tactical decision-making in autonomous systems. *Journal of Systems and Software*, 110:222–238, 2015.
3. Alex M Andrew. Introduction to evolutionary computing. *Kybernetes*, 2013.
4. John R Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.
5. John R. Koza and Riccardo Poli. *Genetic Programming*, pages 127–164. Springer US, Boston, MA, 2005.
6. John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* University Michigan Press, 1975.
7. Wataru Fujishima and Tomoharu Nagao. Genetic matrix algorithm ; simultaneous optimization of structure and numerical parameters. *IEEJ Transactions on Electrical and Electronic Engineering*, 3(1):84–91, 2008.
8. Robert I Mckay, Nguyen Xuan Hoai, Peter Alexander Whigham, Yin Shan, and Michael Oeill. Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines*, 11(3-4):365–396, 2010.
9. Alex Champandard. Behavior trees for next-gen game ai. In *Game Developers Conference, Audio Lecture*, 2007.
10. Teck-Hou Teng, Ah-Hwee Tan, and Loo-Nin Teow. Adaptive computer-generated forces for simulator-based training. *Expert Systems with Applications*, 40(18):7341–7353, 2013.
11. Armon Toubman, Jan Joris Roessingh, Pieter Spronck, Aske Plaat, and H. Jaap van den Herik. Dynamic scripting with team coordination in air combat simulation. In *Modern Advances in Applied Intelligence - 27th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2014, Kaohsiung, Taiwan, June 3-6, 2014, Proceedings, Part I*, pages 440–449, 2014.
12. Rick Evertsz, John Thangarajah, Nitin Yadav, and Thanh Li. Tactics development framework. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1639–1640. International Foundation for Autonomous Agents and Multiagent Systems, 2014.
13. Martin Fowler. *Domain-specific languages*. Pearson Education, 2010.
14. Philip Kerbusch and Paul Eigeman. Flexible and reusable tactical behaviour models for combat aircraft. Technical report, DTIC Document, 2010.
15. Sandeep Mulgund, Karen Harper, and Greg Zacharias. Large-scale air combat tactics optimization using genetic algorithms. *Journal of Guidance, Control, and Dynamics*, 24(1):140–142, 2001.
16. Ko-Hsin Liang and Kuei-Ming Wang. Using simulation and evolutionary algorithms to evaluate the design of mix strategies of decoy and jammers in anti-torpedo tactics. In *Proceedings of the 38th conference on Winter simulation*, pages 1299–1306, 2006.
17. Michael Jay Timmerman. *A genetic algorithm based anti-submarine warfare simulator*. PhD thesis, Mon-

terey, California. Naval Postgraduate School, 1993.
18. Robert E Smith and Bruce A Dike. Learning novel fighter combat maneuver rules via genetic algorithms. *International Journal of Expert Systems*, 8(3):247–276, 1995.
19. Vincent W Porto, Michael Hardt, David B Fogel, Kenneth Kreutz-Delgado, and Lawrence J Fogel. Evolving tactics using levels of intelligence in computer-generated forces. In *AeroSense'99*, pages 262–270. International Society for Optics and Photonics, 1999.
20. Teck-Hou Teng, Ah-Hwee Tan, Yuan-Sin Tan, and Adrian Yeo. Self-organizing neural networks for learning air combat maneuvers. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2012.
21. Pieter Spronck, Marc Ponsen, Ida Sprinkhuizen-Kuyper, and Eric Postma. Adaptive game ai with dynamic scripting. *Machine Learning*, 63(3):217–248, 2006.
22. Armon Toubman, Jan Joris Roessingh, Pieter Spronck, Aske Plaat, and Jaap van den Herik. Improving air-to-air combat behavior through transparent machine learning. In *The Interservice/Industry Training, Simulation & Education Conference (I/ITSEC), Orlando, Florida, USA*, 2014.
23. NATO. Human behavior representation in constructive simulation. Technical Report RTO-TR-HFM-128, NATO RTO, 2009.
24. Yong-lin LEI, Qun LI, Feng YANG, Wei-ping WANG, and Yi-fan ZHU. A composable modeling framework for weapon systems effectiveness simulation. *Systems Engineering Theory & Practice*, 33(11):2954–2966, 2013.
25. Carl Von Clausewitz. *On war*, volume 1. London, N. Trübner & Company, 1873.
26. Robert L Shaw. *Fighter combat: Tactics and maneuvering*. Naval Institute Press, 1985.
27. Ogren Petter. *Increasing Modularity of UAV Control Systems using Computer Game Behavior Trees*. Guidance, Navigation, and Control and Co-located Conferences. American Institute of Aeronautics and Astronautics, 2012.
28. Michele Colledanchise and Petter Ögren. How behavior trees modularize robustness and safety in hybrid systems. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, IL, USA, September 14-18*, pages 1482–1488, 2014.
29. EMF project. `http://www.eclipse.org/modeling/emf/`.
30. GMF project. `http://www.eclipse.org/modeling/gmp/`.
31. Acceleo project. `http://www.eclipse.org/acceleo/`.
32. Jian Yao, Ning Zhu, Junqing Xu, Shuai Chen, and Yonglin Lei. A domain specific language for tactic representation in engagement level simulation. In *30th European Simulation and Modelling Conference, Gran Canaria, Spain*, 2016.
33. Peter Alexander Whigham. *Grammatical bias for evolutionary learning*. PhD thesis, 1996.
34. Adam Nohejl. Grammar-based genetic programming. Master's thesis, Charles University of Prague, 2011.