Simulating a Logistics Enterprise Using an Asymmetrical Wargame Simulation with Soar Reinforcement Learning and Coevolutionary Algorithms

Ying Zhao
Naval Postgraduate School
yzhao@nps.edu

Erik Hemberg
Massachusetts Institute of Technology, CSAIL
hembergerik@csail.mit.edu

Nate Derbinsky
Northeastern University
n.derbinsky@northeastern.edu

Gabino Mata
US Marine Corp.
gabino.mata@usmc.mil

Una-May O'Reilly
Massachusetts Institute of Technology, CSAIL
unamay@csail.mit.edu

ABSTRACT

We demonstrate an innovative framework (CoEvSoarRL) that leverages machine learning algorithms to optimize and simulate a resilient and agile logistics enterprise to improve the readiness and sustainment, as well as reduce the operational risk. The CoEvSoarRL is an asymmetrical wargame simulation that leverages reinforcement learning and coevolutionary algorithms to improve the functions of a total logistics enterprise value chain. We address two of the key challenges: (1) the need to apply holistic prediction, optimization, and wargame simulation to improve the total logistics enterprise readiness; (2) the uncertainty and lack of data which require large-scale systematic what-if scenarios and analysis of alternatives to simulate potential new and unknown situations. Our CoEvSoarRL learns a model of a logistic enterprise environment from historical data with Soar reinforcement learning. Then the Soar model is used to evaluate new decisions and operating conditions. We simulate the logistics enterprise vulnerability (risk) and evolve new and more difficult operating conditions (tests); meanwhile we also coevolve better logistics enterprise decision (solutions) to counter the tests. We present proof-of-concept results from a US Marine Corps maintenance and supply chain data set.

CCS CONCEPTS

• **Security and privacy** → **Access control**; *Vulnerability management*; • **Theory of computation** → *Evolutionary algorithms*;

KEYWORDS

coevolutionary algorithms, reinforcement learning, risk, logistics

ACM Reference Format:
Ying Zhao, Erik Hemberg, Nate Derbinsky, Gabino Mata, and Una-May O'Reilly. 2021. Simulating a Logistics Enterprise Using an Asymmetrical Wargame Simulation with Soar Reinforcement Learning and Coevolutionary Algorithms. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3449726.3463172

GECCO '21 Companion, July 10–14, 2021, Lille, France

ACM ISBN 978-1-4503-8351-6/21/07…$15.00
https://doi.org/10.1145/3449726.3463172

1 INTRODUCTION

A logistics enterprise can contain many business processes in the areas of maintenance, supply, transportation, health services, general engineering, and finance as a whole system. For example, the U.S. Marine Corps (USMC) maintenance and supply chain is a complex enterprise and exemplifies a socio-technological infrastructures that require continuous learning and optimization. Machine learning (ML) algorithms and game theory [9, 14], combined in "deep learning" to perform artificial intelligence (AI) benchmark tasks, demonstrate superb performance in comparison to human [3, 15]. It is imperative for the USMC to adopt more advanced data sciences, including ML and AI techniques, to the end-to-end logistics chain. Specifically, it is important to perform predictive analysis jointly with what-if scenarios and analysis of alternatives to simulate new environmental and operation conditions that the USMC logistics has never encountered before to alleviate operational risk, e.g. critical mission delays. This problem is especially challenging because a USMC expeditionary unit's structure (e.g., table of organization and equipment) may include many parts. For planning the readiness of future USMC missions each part of the expeditionary unit has specific needs for the duration and frequency of manpower, as well as resources required for maintenance. Based on historical data, a predictive model needs to infer the most probable parts based on failure rates, demand history, and available manpower to support the units' operations in normal conditions as well as new, unexpected, and unknown conditions. To compute operation condition perturbations for replenish and maintenance a logistic enterprise simulation needs to create additional conditions, e.g. an unknown blast, desert environment, and corrosion consideration. Since there might be no historical data available for new operation conditions, traditional predictive modeling and simulation analysis might not be directly applicable. In this paper we investigate how to integrate prediction and simulation to modify the logistic enterprise demand models and generate new data based on reinforcement learning [7, 16] and coevolutionary algorithms [6, 12] to reduce operational risk.

The contribution of this paper is to integrate Soar reinforcement learning (Soar-RL) [7] as a predictive model with coevolutionary algorithms [12] to perform prediction, optimization, and wargame simulation in a use case in the USMC maintenance and supply chain logistics enterprise. The integration of Soar-RL with coevolutionary algorithms is modeled as an asymmetrical wargame simulation where a logistician is known as the self-player (defender/solution) for a logistics enterprise and logistics operation conditions is the opponent (attacker/test) for such an enterprise. The adversary of a logistician are the logistic enterprise operating conditions. Note we interchangeably use self-player for defender or solution, and opponent for attacker or test.

Soar-RL is used for both self-player and opponent to learn its own payoff function. Soar-RL learns and modifies the existing knowledge rules from the external environmental reward and penalty. Coevolutionary algorithms take a population-based approach to model the self-player and opponent engagement and use the payoff from the engagement as a fitness indicator. Both Soar-RL and coevolutionary algorithms support the investigation and simulation of complex operation conditions and adversarial dynamics. The framework (CoEvSoarRL) involves making prediction and planning based on the behavior and interaction of a logistics enterprise with some internal decision factors and with some external factors, such as, the operating environment and adversaries.

The paper is organized as follows. Section 2 presents an overview of the CoEvSoarRL. Section 3 describes the background and related work. Section 4 introduces the USMC logistic enterprise use case. Section 5 has the results for the use case. Section 6 discusses the CoEvSoarRL and the results. Section 7 has conclusions and future work.

2 OVERVIEW OF THE COEVSOARRL

In this paper, we investigate the integration of reinforcement learning and coevolutionary algorithms into an asymmetrical wargame simulation CoEvSoarRL. In such a wargame simulation, the self-player simulates decisions (solutions) made to handle the requirements and problems of a logistics enterprise such as a USMC maintenance and supply organization. In our set up, the opponent also simulates environmental and adversarial conditions (tests) of the USMC logistics enterprise. A self-player is a defender (solution) and the opponent is an attacker (test).

The self-player tries to search and optimize decisions for a defined measure of performance, i.e. its own fitness function, for example, minimize time to return an equipment to its original state of readiness. The opponent generates adversarial conditions as states for the self-player to handle. In this sense, the wargame simulation is a form of large-scale what-if analyses to investigate potential new and unknown operation conditions for a logistics enterprise. In the "coevolution" of the wargame simulation the opponent can behave within the following categories:

Case 1 (random): random actions, e.g., weather, fire, pandemic, earthquake, or other environmental uncertain factors.

Case 2 (strategic complement game): the opponent's actions linking to the interest of the self-player, e.g., a new method of transportation of material.

Case 3 (strategic competition game): deliberately adversarial actions minimizing the effect of the self-player's actions, e.g., an adversarial actor in a logistics chain disrupts a normal operation.

In the coevolutionary setting, the self-player and opponent evaluate their success with separate asymmetric objectives represented by the value returned by their fitness functions. The self-player or opponent makes decisions for its own benefit. The fitness function for the opponent in Case 1 (random) can be a probabilistic model of the environment conditions such as in a Monte Carlo simulation, the self-player needs only to act and handle the uncertainty of an opponent. In Case 2 (strategic complement), self-player can optimize its own fitness by leveraging the benefits of strategic complement. In the USMC logistics use case, environmental or adversarial conditions are often in Case 1 and 3, for example, equipment operation conditions can be probabilistically generated as perturbations in Case 1; or deliberately generated disruptions by adversaries as in Case 3 (strategic competition). In this paper, we focus on a use case of a competitive nature of the self-player and opponent, as in Case 3. For a defined fitness function, the self-player tries to maximize the fitness and the opponent tries to minimize the fitness. We use Soar-RL to learn the fitness functions of both players based on their true interaction data and represent them as knowledge rules of Soar-RL. Both players then use coevolutionary algorithms to compete and evolve together.

In CoEvSoarRL, we define the following terms (see Figure 1)

Environment: Observable data that players can use as input for prediction and optimization. The players cannot change the data.

Test: Observable data that a self-player can use as input for prediction and optimization decisions (actions). The self-player cannot change the data. For example, weather or terrain are state variables that an opponent cannot change. The opponent decides on how to construct a test.

Solution: Variables that the self-player are able to decide in order to overcome the tests and improve fitness. The opponent (test) is evaluated on the solution.

Fitness: Desired outcome and performance in the end of a engagement for each player (self-player and opponent). Each player has its own fitness function. The fitness calculation can be different, asymmetrical for both players.

3 BACKGROUND

In this section we present background and related work for Markov Processes, Soar reinforcement learning and coevolutionary algorithms.

3.1 Markov Processes

The states of the adversary can be hidden (e.g., adversarial intent), similar to hidden Markov models (HMM), or partially observable, similar to partially observable Markov decision process (POMDP) [8]. ML algorithms are used to recover hidden states, partially observable states, or non-observable information. Decisions and fitnesss (typically observed at the end of an engagement) need to be optimized .

HMM and PODMP are based on the Markov property. The accumulative reward, or so called Q-value, does not depend on only

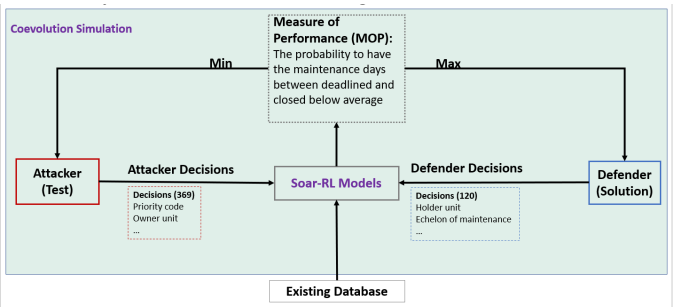


Figure 1: Self-player (Defender) and opponent (Attacker) use opposing fitness functions. Fitness is the probability to have days between deadlined to closed below average. The coevolution simulation generates new data for both self-player and opponent.

one state-action pair, but is computed as an accumulated value of all the states including hidden and non-observable ones up to the current time point. For a finite state set up, the global optimization to maximize the reward of a player has a closed form solution that guarantees the convergence [13, 19]. Therefore, if a planner looks ahead and stops at a specific time point, the last best action is the one that maximizes the estimated Q-value.

The Q-value is the reward in our CoEvSoarRL. In the representation of reinforcement learning and more generic temporal difference learning with relaxed Markov conditions the Q-value can be modeled more flexibly. For example, one can update the Q-value online along the process whenever there is an environment reward observed. We also assume the ground truth fitness can be only observed at the end of a game in our CoEvSoarRL.

3.2 Soar Reinforcement Learning (Soar-RL)

Soar-RL [7] is a cognitive architecture that scalably integrates a rule-based AI system with many other capabilities, including reinforcement learning [16] and long-term memory. The main decision cycle of Soar-RL involves rules that propose new operators (e.g., internal decisions or external actions), as well as preferences for selecting amongst them; an architectural operator-selection process; and application rules that modify agent state. A preference is defined as the probability, contribution, or impact to reach the desired outcome, e.g., the fitness, if an operator (decision) is selected. The reinforcement learning module modifies numeric preferences for selecting operators based on a reward signal, either via internal or external source(s) – importantly, Soar-RL learns in an online, incremental fashion and thus does not require batch processing of (potentially big) data. Soar has been used in modeling large-scale complex cognitive functions for warfighting processes like the ones in a kill chain [18].

In the coevolution simulation played by a self-player and opponent, large collections of asymmetrical decisions and actions for both players to choose. By using Soar-RL, the self-player or opponent take action/state combinations to learn its preference $f_d(D)$ and $f_a(A)$, respectively. D (for self-player) and A (for opponent) could be different and asymmetric for the self-player and opponent.

In a Soar-RL, a preference is defined as the probability of a rule to be used with respect to a total reward (i.e., fitness). To translate into a coevolution simulation, a preference is the contribution of a rule f_k to be selected for a self-player to win. We define preferences $f_k\_v1\_c1$, $f_k\_v0\_c1$, $f_k\_v1\_c0$, and $f_k\_v0\_c0$, where $f_k\_v1\_c1$ means "if a field f_k of either D or A is included ($v = 1$), there is a preference (probability) $f_k\_v1\_c1$ for the self-player to win the game in the end ($c = 1$), i.e., fitness=1 or win the game in the end."

Preferences can be learned from data for the rules. Let m be the number of rules and N the number of data for Soar-RL to perform on-policy learning.

$$Q(s_{t+1}, a_{t+1}) = Q(s_t, a_t) + \alpha[r + \gamma \max_{a \in A} Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (1)$$

Since we only consider an on-policy setting or SARSA, $Q(s_{t+1}, a) = 0$ and let

$$\delta_t = \alpha(r_{t+1} - Q(s_t, a_t)) \quad (2)$$

$\alpha, r_{t+1} = 1$ for a positive reward or -1 for a negative reward. In order to converge, $r_* = Q(s_*, a_*)$ in Equation (2), we ask: Is there a set of preferences $p_1, p_2, ..., p_m$ that makes δ_t in Equation (2) as small as possible when $t \to \infty$.

The total preference is the summation of the preferences from each of the action/state combinations (i.e., Q-value in Equation (1)). For any action/state combination which consists of K fields included ($v = 1$) or not included ($v = 0$). Equation (3) decides a win in the end.

$$\sum_{k=1}^{K} f_k\_v_*\_c1 > \sum_{k=1}^{K} f_k\_v_*\_c0, \quad (3)$$

where $*$ denotes value 1 or 0 for field f_k. The self-player gains a positive reward 1 if a correct action is taken at time t or a negative reward -1 if a wrong action is taken. For example, for an action/state combination, if total preference added for win is larger than lose, the predicted result would be win. If the ground truth is indeed win for this combination for the self-player, then each of the K win rules' preferences related to the combination is modified using a positive reward $\frac{1}{K}$. If the ground truth is lose for this combination, each of the same K rules' preferences is modified using a negative reward $-\frac{1}{K}$. In other words, Soar-RL always modifies the rules that involve the predicted win or loss. Note some fields that are not included ($v = 0$) can also contribute positively to the win ($c = 1$) in Equation (3), this is an example of counterfactuals learning [11] implemented in Soar-RL.

3.3 Coevolutionary Algorithms

Coevolutionary algorithms [6, 10, 12] related to evolutionary algorithms and genetic algorithms [2, 4], explore domains in which the quality of a candidate solution (e.g., an action combination) is determined by its ability to successfully pass some set of tests (attacks), for example, solutions (defenses) in a logistics chain need to pass the known operating conditions that are difficult or adversarial tests (attacks). Competitive coevolutionary algorithms are used to solve minmax-problems similar to those encountered by generative adversarial networks (GANs) [1, 5, 17]. Adversarial engagements of players can be computationally modeled. Competitive coevolutionary algorithms take a population-based (parallel) approach to iterative adversarial engagement and can explore a different behavioral space. The use case tests (adversarial attacker

Algorithm 1 Coevolutionary Algorithm

Input:

T: number of iterations fs, fo: Fitness functions

μ: mutation probability, λ : population size

$\mathbf{A}_0 \leftarrow [\mathbf{a}_{1,0}, \ldots, \mathbf{a}_{\lambda,0}]$ ▷ Initial test population

$\mathbf{D}_0 \leftarrow [\mathbf{d}_{1,0}, \ldots, \mathbf{d}_{\lambda,0}]$ ▷ Initial solution population

$t \leftarrow 0$ ▷ Initialize iteration counter

repeat

  $t \leftarrow t + 1$ ▷ Increase counter

  $\mathbf{A}_t \leftarrow select(\mathbf{A}_{t-1}))$ ▷ Selection

  $\mathbf{A}_t \leftarrow variation(\mathbf{A}_t, \mu))$ ▷ Mutation and crossover

  ▷ Best test

  $\mathbf{a}'_*, \mathbf{d}'_* \leftarrow \arg\min_{\mathbf{a}\in\mathbf{A}_t} \arg\max_{\mathbf{d}\in\mathbf{D}_{t-1}} fo(\mathbf{a}, \mathbf{d})$

  ▷ Replace worst test

  if $fo(\mathbf{a}'_*, \mathbf{d}'_*) < fo(\mathbf{a}_{\lambda,t-1}, \mathbf{d}_{\lambda,t-1})$ **then**

    $\mathbf{a}_{\lambda,t-1} \leftarrow \mathbf{a}'_*$ ▷ Update population

  $\mathbf{A}_t \leftarrow \mathbf{A}_{t-1}$ ▷ Copy population

  $t \leftarrow t + 1$ ▷ Increase counter before alternating to solutions

  $\mathbf{D}_t \leftarrow select(\mathbf{D}_{t-1}))$ ▷ Selection

  $\mathbf{D}_t \leftarrow variation(\mathbf{D}_t, \mu))$ ▷ Mutation and crossover

  ▷ Best solution

  $\mathbf{a}'_0, \mathbf{d}'_0 \leftarrow \arg\min_{\mathbf{a}\in\mathbf{A}_t} \arg\max_{\mathbf{d}\in\mathbf{D}_t} fs(\mathbf{a}, \mathbf{d})$

  ▷ Replace worst solution

  if $fs(\mathbf{a}'_0, \mathbf{d}'_0) > fs(\mathbf{a}_{\lambda,t}, \mathbf{d}_{\lambda,t-1})$ **then**

    $\mathbf{d}_{\lambda,t-1} \leftarrow \mathbf{d}'_0$ ▷ Update population

  $\mathbf{D}_t \leftarrow \mathbf{D}_{t-1}$ ▷ Copy population

until $t \geq T$

$\mathbf{a}_*, \mathbf{d}_* \leftarrow \arg\min_{\mathbf{a}\in\mathbf{A}_T} \arg\max_{\mathbf{d}\in\mathbf{D}_T} fs(\mathbf{a}, \mathbf{d})$ ▷ Best test

return $\mathbf{a}_*, \mathbf{d}_*$

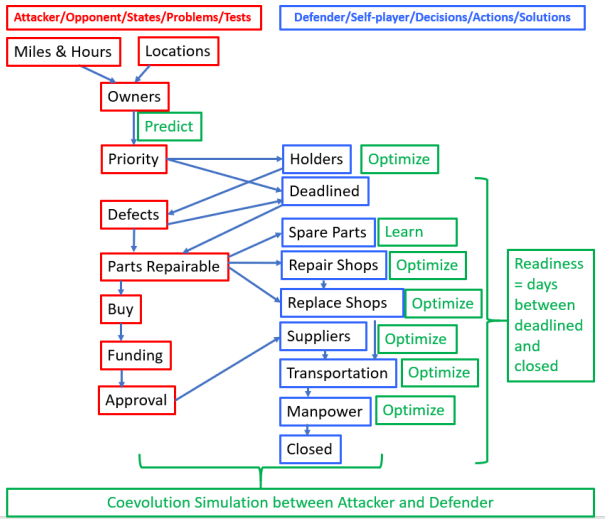


Figure 2: An example of USMC logistic enterprise equipment maintenance and logistics process

population) are actively or passively thwarting the effectiveness of the problem solution (defender). The coevolutionary algorithms are used to identify successful, novel, as well as the most effective means of solutions (defenses) against various tests (attacks). In this competitive game, the test (attacker) and solution (defender) strategies can lead to an arms race between the adversaries, both adapting or evolving while pursuing conflicting objectives.

A basic coevolutionary algorithm evolves two populations with e.g. tournament selection and for variation uses crossover and mutation. One population comprises tests (attacks) and the other solutions (defenses). In each generation, engagements are formed by pairing attack and defense. The populations are evolved in alternating steps: first the test population is selected, varied, updated and evaluated against the solutions, and then the solutions population is selected, varied, updated and evaluated against the tests. Each test–solution pair is dispatched to the engagement component and the result is used as a part of the fitness for each of them. Fitness is calculated over all an adversary's engagements.

4 USE CASE - COEVSOARRL FOR A USMC LOGISTICS ENTERPRISE

As an example use case we use a USMC maintenance and supply chain data set. Some operational cost and risk comes from the uncertainty and unknown operation conditions that constitute the key challenges for the USMC maintenance and supply chain. For example, the uncertainty of the reliability of assets has caused the USMC to maintain and operate with excess equipment and supplies. In the past, data analytics including ML have been used by the USMC to address variety of challenges. For example, predictive models have been used to predict equipment reliability and probability of failure, in order to infer the numbers of spare parts required to improve stock performance and synchronize budget execution.

Our CoEvSoarRL addresses prediction and optimization, more importantly, large-scale systematic what-if scenarios and analysis of alternatives and wargame simulation towards discovering vulnerability and optimizing a total readiness for the maintenance and supply chain for new and unseen operation conditions. We now describe the the USMC logistics enterprise, the CoEvSoarRL and the setup.

4.1 Logistics Enterprise

Currently in the USMC maintenance and supply system, as shown in Figure 2, a major USMC equipment or part of an equipment fails, a service ticket is opened. A service ticket is then taken care of in a long sequence of actions (decisions) such as to be repaired, replaced (in various exchanges and echelons) or requisitioned from the supply system. If an equipment or related parts cannot be repaired or replaced such as a consumable like fuel or battery, they are purchased or requisitioned. The service ticket is *closed* when the equipment and parts are returned and ready to use. The time (usually days) between the open date and close date of the service ticket indicates how long it takes to make equipment and associated parts ready to use again through the maintenance and supply chain. As shown in Figure 2, for major USMC equipment, before its owner starts a service requirement, how the equipment is used such as miles, hours, locations, and missions might decide the service priority of the equipment such as urgent, critical, deadlined, routine maintenance etc. When equipment is *deadlined* it means that the equipment is not operational and it stops the whole mission, i.e. poses a risk to the mission, e.g. a faulty seat belt can make equipment deadlined. Sometimes parts are repairable like an engine or consumable like fuel.

4.2 CoEvSoarRL

For this problem, coevolutionary algorithms are used with the following set up in a coevolution simulation:

Opponent (Test): The context (conditions) of the USMCS logistics enterprise. Logistics chain problems with action (decision) variables A such as owner's profile, equipment usage (miles and hours, operation locations), service priority, defect codes, which part is repairable, what part is to buy, which is the funding level for purchase, what is the approval status.

Self-player (Solution): The logistician (demander) in the USMC logistics enterprise. Logistics chain solutions with action (decision) variables D such as holders (who perform initial diagnosis of defects), how many spare parts, which repair or replace shops to use, which suppliers or transportation methods to use, what manpower to use. These data fields are decisions and actions that can be taken to pass the tests and optimize the measure of performance (fitness).

Fitness: The probability to have the maintenance days between deadlined and closed is less than the average computed from the historical data set.

We use a machine learning algorithm such as a predictive model or reinforcement learning model like Soar-RL is to learn a fitness f between test, solution, and fitness i.e., $f_d(D)$ and $f_a(A)$. For coevolution we use a wargame simulation. The test (opponent) needs to minimize the fitness and the solution (self-player) needs to maximize the fitness. In the USMC logistic enterprise use case, the test and solution have opposite, not necessary zero-sum, optimization objectives f_d and f_a, their action (decision) variables are different so the wargame simulation is asymmetrical.

In the coevolutionary search, the opponent (test) needs to evolve logistics contexts described by the combinations of the opponent's decision variables A to decrease the fitness of the self-player (solution); The self-player (solution) needs to evolve logistics decisions described by the combinations of the self-player's decision variables D to increase the fitness. A high fitness for the self-player means better readiness for the USMC logistics. Thus if the self-player (solution) can handle more difficult contexts generated by the opponent (test), the readiness is improved. As shown in Figure 1 both the Opponent and Self-Player evolve and coevolve and both are guided by the fitness functions that reflect the adversaries competing objectives. We use Soar-RL to learn two fitness functions separately for the Self-Player and Opponent and then use Soar-RL models jointly in the coevolutionary algorithms. The coevolution simulation generates data that does not exist in the historical database.

4.3 Setup

We first fuse data for a type of USMC equipment from a few databases including maintenance, supply, and equipment usage. We then aggregate the data for each service ticket with the following sources:

Maintenance history: unique number of service request types, unique number of defect codes, unique number of operational status, unique number of echelon of maintenance, unique number of master priority code, count of job status dates, count of service cross-references, unique number of service parts, count of service activities, count of task numbers.

attacker fitness: -0.340177344543229,
attacker's configuration:
<(O)MASTER_PRIORITY_CODE_06BUrgent>
<(O)SERVICE_REQUEST_TYPE_MaintenanceCM>
<(O)OWNER_UNIT_ADDRESS_CODE_M11700>
<(O)EQUIP_OPER_TIME_CODE_EOTCHhours>
<(O)DEFECT_CODE_ENGINOP>
<(O)OPERATIONAL_STATUS_Deadlined>

defender's configuration:
<(S)HOLD_UNIT_IDENT_CODE_M14030>
<(S)MIMMS_RAC_MIM004>
<(S)ECHELON_OF_MAINT_2>
<(S)JOB_STATUS_CODE_Closed>

Figure 3: An example of opponent with id $10fe75$ and self-player with id $b642cf$ configurations exists in the historical database, i.e., it is one of the 1,212 service tickets. It is a configuration for opponent and self-player with the fitness value -0.340 in Figure 7(a). It is the starting point for the coevolution.

Table 1: Coevolutionary Algorithm parameter settings.

Parameter	Value
Population size	10
Max length	489
Elite size	1
Generations	80
Tournament size	2
Crossover probability	0.8 (single point crossover)
Mutation probability	0.1 (int flip mutation)
Runs	30

Requisition data: maximum of part charge, count of document numbers, count of parts update dates, count of requirement numbers, count of unit issue, count of item types, count of supply route locations.

Equipment usage data: owner unit address code, equipment operation time code, and meter reading.

A total of 489 aggregated variables represent states and decisions for both the self-player and opponent. We divide the 489 variables into two groups: the Opponent has 369 variables labeled "(O)" and Self-Player has 120 variables labeled "(S)". They all potentially correlate with the fitness. The sample data set contains 1,212 service numbers (tickets) that are deadlined for about two years for the type of equipment, 132 tickets (11%) have the days between deadlined and closed date more than 32 days (32 days is the mean of the days between the deadlined and closed dates for the data set). The number of rules learned from the data set using Soar-RL for the fitness functions $f_d(D)$ and $f_a(A)$ are 489*4=1,956, respectively.

Figure 3 shows an example of opponent (maintenance problem with variables with "(O)") and self-player (maintenance solution with variables with "(S)") configurations which exist in the historical database.

Table 1 show the parameters used for the coevolutionary algorithm runs. We used a version of Donkey GE https://github.com/flexgp/donkey_ge with each player decision encoded as true or false.

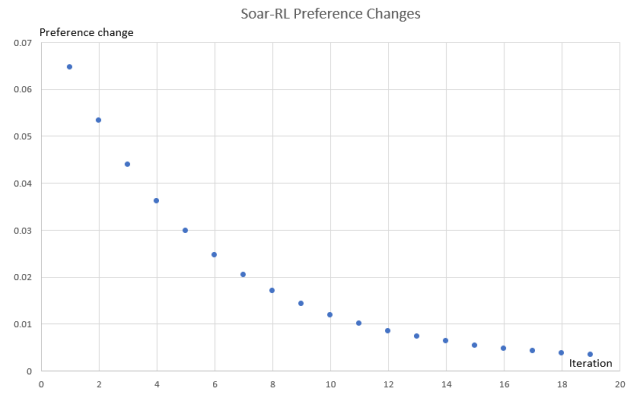


Figure 4: The convergence of learning preferences of the rules in Soar-RL

5 RESULTS FOR USMC LOGISTICS ENTERPRISE WITH COEVSOARRL

Learning Soar-RL rules. Since Soar-RL is an online on-policy machine learning algorithm, it is important to show the algorithm converges in practice. Figure 4 shows the convergence of the the preference change for the USMC use case when learning the self-player's fitness, i.e., $f_d(D)$ when the iteration is 20. Learning of the opponent's fitness $f_a(A)$ has similar convergence, each generates 1,956 Soar-RL rules.

Coevolving self-player and opponent. Figures 7(a), (b), and (c) show the early generations of the evolving Opponents and how their fitness values (the values on the heatmaps) change in each generation. The generation 0 in Figure 7(a) starts with the existing data as shown in Figure 3, which is the least fit logistics test against the most fit logistics solution ($b642cf$). The Opponent evolves three more fit tests $8f5b7a$, $2c4081$, and $c1d409$ with the fitness value -0.340,-0.339, and -0.339 respectively. Figure 5 shows $c1d409$ is one of the more fit tests that do not exist in the historical USMC logistics enetrprise database. This indicates the coevolutionary algorithms are capable of performing a what-if scenarios and analysis of alternatives that reveal possible new logistics tests. The wargame simulation also suggests various self-player solutions to handle the new opponent(test).

Figure 7(b) shows the evolution of the Opponent in generation 1. The Opponent evolves into more fit ones such as $d284e4$ while the Self-Player stays the same. Figure 7(c) shows the Opponent continuously evolves to more fit ones and the Self-Player evolves too, however, can not surpass the fitness of the best one $b642cf$.

Similarly, Figures 7(d), (e), and (f) show the coevolution search process of the Self-Player. We see the same patterns here: When the Opponent evolves into more fit ones such as $d284e4$ while the Self-Player evolves in Figure 7(e) and Figure 7(f) as well, however, the Self-Player does not evolve to the ones that have higher fitness than the one of $b642cf$ (best fitness=0.204).

attacker fitness: -0.3393015847887526,
attacker's configuration:
<(O)MASTER_PRIORITY_CODE_03ACritical>
<(O)SERVICE_REQUEST_TYPE_MaintenanceCM>
<(O)OWNER_UNIT_ADDRESS_CODE_M14030>
<(O)EQUIP_OPER_TIME_CODE_EOTCHhours>
<(O)DEFECT_CODE_FCONSAFDL>
<(O)OPERATIONAL_STATUS_Deadlined>

defender's configuration:
<(S)HOLD_UNIT_IDENT_CODE_M14030>
<(S)MIMMS_RAC_MIM004>
<(S)ECHELON_OF_MAINT_2>
<(S)JOB_STATUS_CODE_Closed>

Figure 5: An example of opponent (attacker) starts with a test id $10fe75$ that exists in the historical database and evolves to a test id $c1d409$ which does not exist in the historical database, i.e., it is not one of the 1,212 service tickets. It is an evolved test with the fitness value -0.339 in Figure 7(a) compared to the fitness of 0.340 of $10fe75$. The master priority code, service type, owner and defect combination could make the test fitter (harder). The simulation suggests a self-player solution for this opponent is $b642cf$.

attacker fitness: -0.1961503845037802,
attacker's configuration:
<(O)MASTER_PRIORITY_CODE_02ACritical>
<(O)MASTER_PRIORITY_CODE_13CRoutine>
<(O)MASTER_PRIORITY_CODE_06BUrgent>
<(O)MASTER_PRIORITY_CODE_05BUrgent>
<(O)SERVICE_REQUEST_TYPE_Service>
<(O)SERVICE_REQUEST_TYPE_MaintenanceMOD>
<(O)SERVICE_REQUEST_TYPE_MaintenanceSL3>
<(O)OWNER_UNIT_ADDRESS_CODE_M20450>
<(O)OWNER_UNIT_ADDRESS_CODE_M12120>

...

<(O)DEFECT_CODE_COMP>
<(O)DEFECT_CODE_ENGSHORT>
<(O)DEFECT_CODE_FUELCBB>
<(O)OPERATIONAL_STATUS_OperationalMinor>
<(O)OPERATIONAL_STATUS_Deadlined>

defender's configuration:
<(S)HOLD_UNIT_IDENT_CODE_M14030>
<(S)HOLD_UNIT_IDENT_CODE_M14033>
<(S)HOLD_UNIT_IDENT_CODE_MMV444>
<(S)HOLD_UNIT_IDENT_CODE_M20450>
<(S)SERVICE_ACTIVITY_GCSSMCLabor>
<(S)SSC_RIC_SMS>
<(S)JOB_STATUS_CODE_RPRPRGS>
<(S)JOB_STATUS_CODE_AWTGContactTeam>
<(S)JOB_STATUS_CODE_COMPEVAC>
<(S)JOB_STATUS_CODE_PendingClosure>

Figure 6: An example of opponent continues evolving to $d284e4$ in generation 2 in Figure 7(c) with self-player $257fa4$, fitness -0.196. These evolved opponents are not in the existing database or one of the 1,212 service tickets. The tests contain more service types, owner units and defect codes which make opponent more difficult for the self-player to handle.

6 DISCUSSION

Figures 8(a) and (b) show the Opponent's and Self-Player's mean and best fitness values changing sharply for the first three generations in the coevolutionary algorithms, respectively. The trends validate the results and analyses that the Self-Player, representing the logistics solutions, gets worse on average while the Opponent,

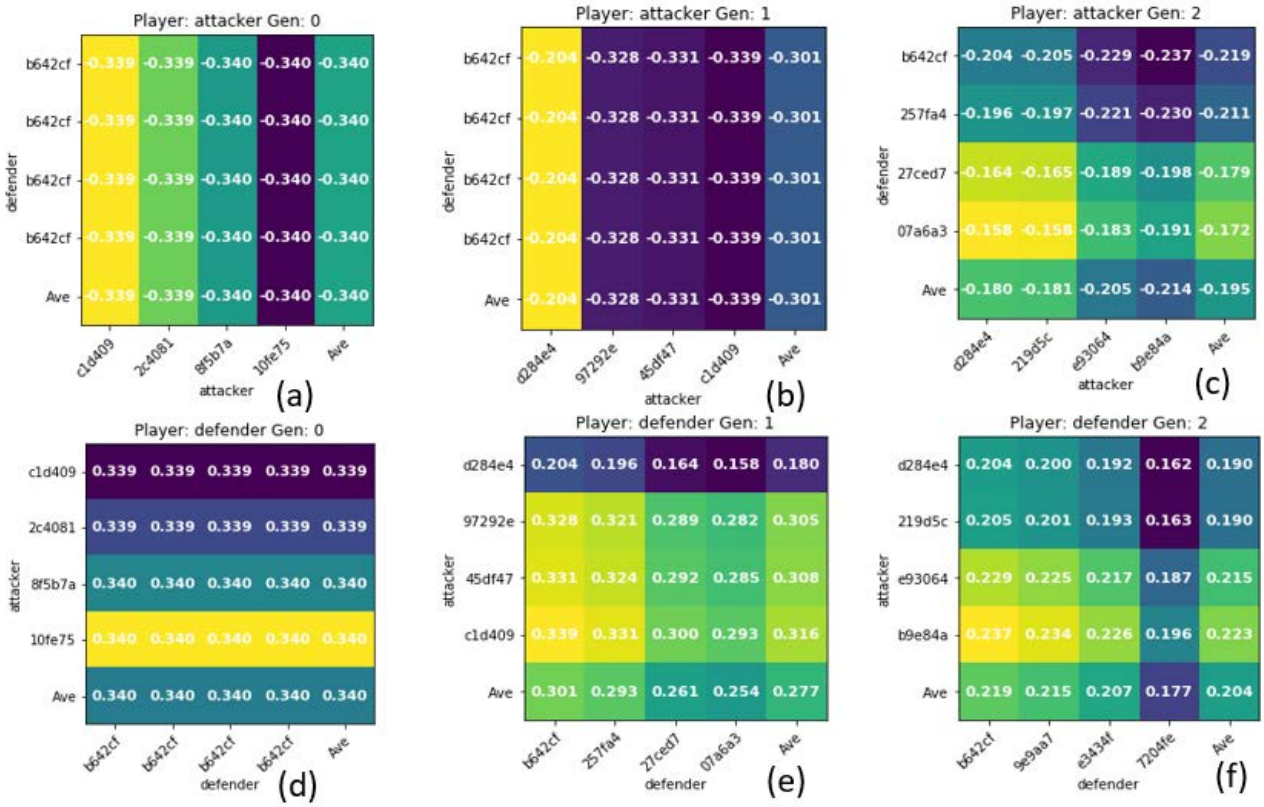


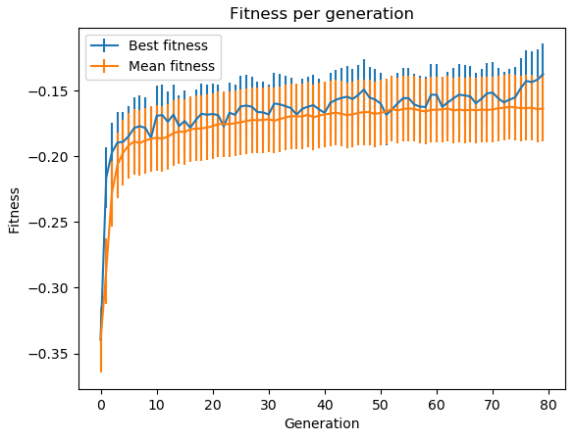
Figure 7: Opponent and Self-Player's evolutions of three generations. (a) opponent starts with a test id $10fe75$ that exists in the historical database and evolves to a test id $c1d409$ which does not exist in the historical database. Opponent continues evolving to $d284e4$, which is more fit than $c1d409$ in generation 1 in Figure 7(b) with self-player $b642cf$, and fitness -0.204; in generation 2 in Figure 7(c) with self-player $257fa4$, fitness -0.196. These evolved opponents are not in the existing database or one of the 1,212 service tickets. Notice that the self-player does not evolve to any better solution than $b642cf$ as shown in Figure 7(d) to (f). The tests contain more service types, owner units and defect codes which are less feasible in a normal time, however, more feasible in a time of conflict and disruption, and can put challenge the solution side of the logistics chain.

representing logistics tests, gets better on average in the coevolution simulation. The algorithms simulate and discover possible new tests that might need new solutions to handle. A "risk" can be discovered when a solution can not evolve to be better solutions that can handle the new tests. The example of evolved tests in Figure 5 is new and more difficult tests that do not exist in the historical database. The one in Figure 5 in is more feasible than some others, it could present challenges to the current logistics solution process.

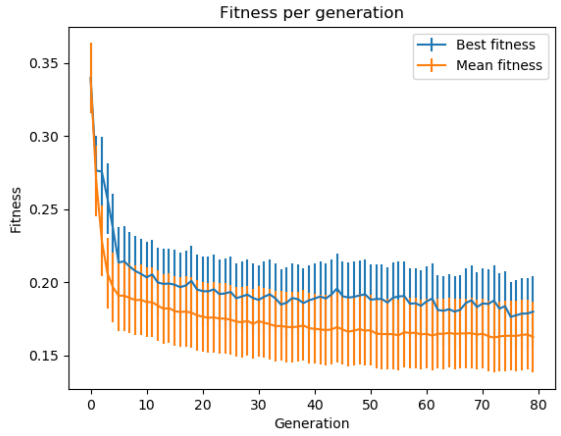
Why use Soar-RL and Coevolutionary Algorithms. Soar-RL, as an reinforcement learning algorithm, is rule-based and explainable, as well as flexible enough to include and modify rules of engagement, knowledge, and tactics, and also discover new rules from data such as 1,956 rules generated in our data set. Soar-RL can also perform online and on-policy learning, these characteristics makes it a applicable for defense application. In this paper, we also combine Soar-RL and coevolutionary algorithms so it can be used in alternating optimization and wargame simulation of two competitive players. Soar-RL allows coevolutionary algorithms to search and optimize the logistics solutions and total readiness to simulate and handle difficult, new, and unknown environmental and operational conditions.

In our CoEvSoarRL, a Soar-RL model is trained with sequences (time-series) of states and actions combinations reinforced with the final fitness results. The trained Soar-RL models are then used to guide the search and evolution process of coevolutionary algorithms in any time point by pulling out the rules usable only up to that time point. For example, for a new opponent or logistic test that is never seen before or overwhelmingly challenging, one can simulate better solutions by adding rules of new possible decisions and actions with estimated preferences from domain experts.

The mutation and crossover evolutionary operators are unsupervised, and can produce better individuals (solutions or tests). Only the selection operator for coevolutionary algorithms requires a guidance of fitness function, which can be implemented using a supervised or reinforcement machine learning algorithm, in our case,



(a) Opponent



(b) Self-Player

Figure 8: Opponent and Self-Player's mean and best fitness values in three generations:The wargame simulation can systematically simulate and discover possible new tests or "vulnerabilities (risks)" for the logistics system and evolve solutions accordingly.

the Soar-RL. Some data is required for training the coevolutionary algorithm used.

Feasibility of Tests and Solutions. The total number of decision combinations for the opponent (tests) is 2^{369}, and total number of decision combinations for self-player (solution) is 2^{120}. However, not all the solutions or tests are feasible in real life. The tests (opponents) and solutions (self-players) often have to comply to certain feasibility constraints. It is possible to show that our CoEvSoarRL can use the association patterns in the search space, consequently, reduce the data sample size required to train machine learning models. This will be future research of context-dependent Soar-RL and coevolutionary algorithms.

7 CONCLUSION AND FUTURE WORK

We demonstrate a proof-of-concept framework (CoEvSoarRL) using Soar-RL based fitness prediction jointly with coevolution algorithms that apply to a USMC logistics data set and beyond. We demonstrate the CoEvSoarRL capable of performing what-if analysis that reveal new logistics tests and solutions, and possible risks in a logistics system. The simulation also can also suggest novel and fitter solution (self-player) decisions to handle new tests (opponent) that have never seen before. We show the CoEvSoarRL provides an innovative wargame simulation to improve total readiness of a resilient and agile logistics enterprise.

Future work will look at providing context-dependent information to improve the feasibility of solutions. We will also investigate the parameter sensitivity of the CoEvSoarRL.

ACKNOWLEDGMENTS

Authors would like to thank the Naval Postgraduate School (NPS)'s Naval Research Program (NRP) and the Office of Naval Research (ONR)'s Naval Enterprise Partnership Teaming with Universities for National Excellence (NEPTUNE 2.0) program for supporting the research. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied of the U.S. Government.

Research was sponsored by the United States Air Force Research Laboratory and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

REFERENCES

[1] Sanjeev Arora, Rong Ge, Yingyu Liang, Tengyu Ma, and Yi Zhang. 2017. Generalization and equilibrium in generative adversarial nets (gans). In *International Conference on Machine Learning*. PMLR, 224–232.
[2] Thomas Back. 1996. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press.
[3] Noam Brown and Tuomas Sandholm. 2017. Safe and nested subgame solving for imperfect-information games. *arXiv preprint arXiv:1705.02955* (2017).
[4] David E Goldberg. 1989. Genetic algorithms in search. *Optimization, and MachineLearning* (1989).
[5] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial networks. *arXiv preprint arXiv:1406.2661* (2014).
[6] Krzysztof Krawiec and Malcolm Heywood. 2019. Solving complex problems with coevolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 975–1001.
[7] John E Laird. 2019. *The Soar cognitive architecture*. MIT press.
[8] John Loch and Satinder P Singh. 1998. Using Eligibility Traces to Find the Best Memoryless Policy in Partially Observable Markov Decision Processes.. In *ICML*. 323–331.
[9] John F Nash et al. 1950. Equilibrium points in n-person games. *Proceedings of the national academy of sciences* 36, 1 (1950), 48–49.
[10] Una-May O'Reilly, Jamal Toutouh, Marcos Pertierra, Daniel Prado Sanchez, Dennis Garcia, Anthony Erb Luogo, Jonathan Kelly, and Erik Hemberg. 2020. Adversarial genetic programming for cyber security: A rising application domain where GP matters. *Genetic Programming and Evolvable Machines* 21, 1 (2020), 219–250.
[11] Judea Pearl and Dana Mackenzie. 2018. *The book of why: the new science of cause and effect*. Basic books.

[12] Elena Popovici, Anthony Bucci, R Paul Wiegand, and Edwin D De Jong. 2012. Coevolutionary Principles.
[13] Lawrence R Rabiner. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE* 77, 2 (1989), 257–286.
[14] Eric Rasmusen. 1989. Games and information: An introduction to game theory.
[15] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *nature* 550, 7676 (2017), 354–359.
[16] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.
[17] Jamal Toutouh, Erik Hemberg, and Una-May O'Reilly. 2019. Spatial evolutionary generative adversarial networks. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 472–480.
[18] Ying Zhao, Nate Derbinsky, Lydia Y Wong, Jonah Sonnenshein, and Tony Kendall. 2018. Continual and Real-time Learning for Modeling Combat Identification in a Tactical Environment. In *Proceedings of the NIPS 2018 Workshop on Continual Learning*.
[19] Eric A Zilli and Michael E Hasselmo. 2008. The influence of Markov decision process structure on the possible strategic use of working memory and episodic memory. *PloS one* 3, 7 (2008), e2756.