JOURNAL OF AEROSPACE INFORMATION SYSTEMS
Vol. 20, No. 7, July 2023

Swarm Intelligence in Cooperative Environments: *n*-Step Dynamic Tree Search Algorithm Overview

Marc Espinós Longa,* Antonios Tsourdos,† and Gokhan Inalhan‡
Cranfield University, Cranfield, England MK43 0AL United Kingdom

https://doi.org/10.2514/1.I011086

Reinforcement learning tree-based planning methods have been gaining popularity in the last few years due to their success in single-agent domains, where a perfect simulator model is available: for example, Go and chess strategic board games. This paper pretends to extend tree search algorithms to the multiagent setting in a decentralized structure, dealing with scalability issues and exponential growth of computational resources. The *n*-step dynamic tree search combines forward planning and direct temporal-difference updates, outperforming markedly conventional tabular algorithms such as *Q* learning and state-action-reward-state-action (SARSA). Future state transitions and rewards are predicted with a model built and learned from real interactions between agents and the environment. This paper analyzes the developed algorithm in the hunter–pursuit cooperative game against stochastic and intelligent evaders. The *n*-step dynamic tree search aims to adapt single-agent tree search learning methods to the multiagent boundaries and is demonstrated to be a remarkable advance as compared to conventional temporal-difference techniques.

Nomenclature

A_t	=	joint agent actions at time step t
G	=	return
n	=	number of agents
t	=	current time step
Q^i	=	state–action value table of intelligent agent i
q	=	Bellman state–action value function (expected return for a given state and action)
R_t	=	joint rewards for each learning agent at time step t
S_t	=	joint Markov decision process states at time step t
T	=	terminal time step
α	=	learning rate (step size parameter)
β_m	=	mean moment update parameter
β_v	=	second moment update parameter
γ	=	discount factor
ϵ	=	exploratory parameter of ϵ-greedy action selection policy
n_c	=	pursuers involved in a cooperative capture
λ	=	weight decay
π	=	policy (agent behavior)

I. Introduction

THE *machine learning* field has been expanding over the last decades because of its wide range of application. Typically, a learning process can be classified into three main groups: supervised [1,2], unsupervised [3,4], and reward-based learning methods [5]. Addressing single-agent and multiagent domains often entails partial or no information about the boundary conditions and a high degree of uncertainty. Thus, the problem becomes untreatable through input/output-driven data structures. Moreover, the inherent complexity demands feedback to solve and optimize the problem; i.e., unsupervised techniques are usually not enough. Reward-based learning has demonstrated to be a natural fit in this area due to the capability of generating unique solutions with reinforcements and data restrictions.

Under the reward-based umbrella, *stochastic search* directly learns policies without appealing value functions (i.e., estimates of successor states or state–action pairs), which is an efficient technique for small state spaces. Darwinian models of evolution are used to refine populations of candidate behaviors in evolutionary computation. *Genetic programming* [6] and *coevolutionary algorithms* [7–9] are solid contenders within this field. On the other side, *reinforcement learning* (RL) uses value functions under a *Markov decision process* (MDP) framework by considering every state and action taken at every iteration step. This fact often leads to rigorous policies that, despite the increasing demand of computational resources, achieve greater performance in large search spaces.

When training agents under an MDP, the RL spectrum encompasses many design aspects that can be explored. From one-step updates in sample *temporal-difference* (TD) methods [10–13] to *n*-step algorithms that consider a variable or fixed number of MDP transitions when computing state–action values [14,15]. As a result of bootstrapping (estimates as a function of future value estimates), TD strategies provide faster learning rates at the penalty of higher bias; i.e., more error is induced indirectly on every estimate. In contrast, Monte Carlo methods, which are the most extreme form of *n*-step algorithms, delay updates until the end of the episode (end of a finite sequence of time steps), removing bias but significantly increasing the variance of the optimization process. Both TD and Monte Carlo techniques directly impact the length (depth) of the update rule, as Fig. 1 shows. Otherwise, we can improve the accuracy of our estimates by considering not only the sampled RL transitions but also other potential cases: that is, increasing the width of the update (see Fig. 1). Figure 1 expected updates [16] contemplate all potential trajectories, weighing the value estimates at a higher computational cost.

Planning is another powerful tool for speeding up learning by alternating direct RL updates from real-world interactions with simulated trajectories (model-based methods), which can be derived from agent experience using replay buffers [18,19] or already-built environment models [20]. After defeating the 18-time world champion Go player in 2016 [21], AlphaGo, which is based on the *Monte Carlo tree search* (MCTS) [17], gives a new perspective on tree search single-agent methods. Forward planning through an existing model is used to make predictions and optimize action selection mechanisms. This algorithm has recently inspired many powerful algorithms such as *MuZero* (developed by DeepMind), which was validated and tested with superhuman performance across a large variety of Atari games [22].

Transitioning from single-agent to multiagent frameworks is no trivial task. Even from a *team learning* perspective [5], where agents are treated as a master individual and single-agent techniques apply, there is an exponential computational demand linked to the

Presented as Paper 2022-1839 at the 2022 AIAA SciTech and 2022 American Control Conference, San Diego, CA and Atlanta, GA, January 3–7 and June 8–10, 2022; received 16 December 2021; revision received 17 February 2023; accepted for publication 24 March 2023; published online Open Access 28 April 2023.

*Ph.D. Researcher, School of Aerospace, Transport and Manufacturing, Bedfordshire.

†Head of Center, Director of Research, School of Aerospace, Transport and Manufacturing, Bedfordshire. Senior Member AIAA.

‡Deputy Head of Center, School of Aerospace, Transport and Manufacturing, Bedfordshire. Associate Fellow AIAA.

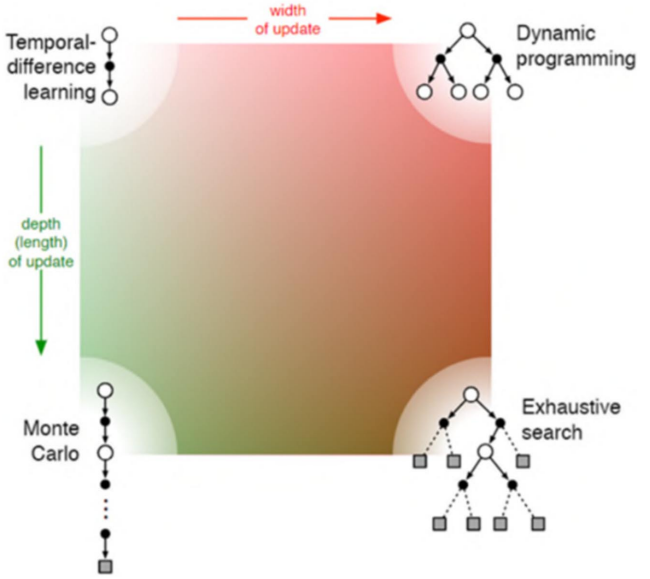


Fig. 1 Simplified map of sample-based learning methods [17].

associated state space (curse of dimensionality problem). Furthermore, like any other centralized system, the swarm depends on a central (leading) node that is a single point of failure to the system [23]. Notice that, herein, we define a swarm as a collection or team of agents that collaborate to achieve a certain goal. On the other hand, *concurrent learning* [5] uses a distributed network of agents to handle multiple learning processes simultaneously [24,25]. Thus, decentralized agent coadaption is required to achieve an optimal swarm behavior; i.e., each learner must modify its behavior according to other coadapting learners. The reward function plays a major role when solving the *multiagent reinforcement learning* (MARL) problem. Nonetheless, distributed systems struggle to find an optimal reward function that maximizes swarm performance due to the existing independent learning processes (also known as the credit assignment problem). Similarly, communication is crucial to enhance the learning process. Although some techniques use direct communication to share TD updates, policies, or even full episodes [26], others opt for indirect bioinspired mechanisms [27,28].

This study aims to apply single-agent tree search RL algorithms to the multiagent domain as a decentralized system through a hybrid reward function, i.e., a combination of team and individual rewards. Inspired by the MCTS and n-step tree backup updates [17], the proposed n-step *dynamic tree search* (NSDTS) algorithm combines forward planning with direct RL, enhancing learning speed and team behavior after algorithm convergence. Action selection mechanisms are improved by using a neural network model, built from real-world experience, to perform a forward tree search. To guarantee system robustness, communication has been disregarded and left for future work; thence, individuals assume the *best play* of agents to carry out future predictions. Lastly, expected updates are used to weigh state–action value estimates that are kept in individual *q tables* (tabular setting). Hence, the presented method addresses, from a theoretical perspective, the fundamentals of a tree search in MARL.

Harvesting previous work [29,30], the developed algorithm is tested and validated in the hunter–prey pursuit environment against Q learning: expected SARSA and SARSA temporal-difference methods. The pursuit game, which involves several learning agents cooperating to capture one or more evaders, has been used in MARL nearly since inception [26,31–36]. This work conceives two scenarios, including a grid world with two learning hunters. The first case considers a stochastic prey that selects actions arbitrarily, and the latter considers an intelligent agent that escapes from chasers given their position.

The rest of the paper is structured as follows: the theoretical foundation in MARL is provided in Sec. II, including MDPs, TD methods, and the NSDTS algorithm. Section III presents an overview of the established system and a description of the tree search models. Section IV formulates the hunter–prey pursuit problem by considering environment features, game rules, and reward function. The results and discussion of the hyperparameter analysis and algorithm performance experimentation are presented in Sec. V. Concluding marks are given in Sec. VI. Future work includes transitioning from tabular to function approximation domain to scale up the experiment, adding complexity with more agents and continuous environments (e.g., real-case applications such as drone delivery services, safe and rescue operations, surveillance, warehousing, etc.). Communication and reward optimization may as well be considered.

II. Theoretical Background

The NSDTS and TD methods comprise common RL elements and are built in an MDP framework. This section offers a brief introduction to RL, including concepts such as MDP, return, or state–action value estimates. Thereafter, the n-step dynamic tree search algorithm is addressed.

A. Markov Decision Process

MDPs are mathematical formalisms that codify the problem of one or more agents interacting with the environment. Nonetheless, there are minor differences between single-agent and multiagent frames. In the MARL configuration, for a given time t and a set of n agents in $S_t = (S_t^0, S_t^1, \ldots, S_t^{n-2}, S_t^{n-1})$ joint states, after executing $A_t = (A_t^0, A_t^1, \ldots, A_t^{n-2}, A_t^{n-1})$ actions through some action selection mechanism, the environment transitions all agents to the next MDP state of $S_{t+1} = (S_{t+1}^0, S_{t+1}^1, \ldots, S_{t+1}^{n-2}, S_{t+1}^{n-1})$ and gives R_{t+1} joint rewards to every learning agent. Each cycle represents an MDP step, and episodes are a collection of step sequences such as $\langle S_0, A_0, R_1, S_1, A_1, \ldots, R_{T-1}, S_{T-1}\rangle$ that, at time $t = T$, reach a terminal joint state.

The goal of RL agents is to maximize long-term rewards. Thus, a balance must be struck between instant gratification and protracted implications of acts. A common approach to solving this dilemma is to use the expected discounted sum of future rewards [Eq. (1)], which is also referred to as return:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{1}$$

with G_t expected to return at time step t, and γ being a discount factor affecting future terms. It is worth noting that Eq. (1) is a finite geometric progression if $0 \leq \gamma \leq 1$.

The MDP features are defined by problem conditions. Sensor constraints, environmental circumstances, or simply experimental settings prevent agents from fully noticing joint states in *partially observable Markov decision processes* [26]. *Factored* MDPs promote parallel processing via state aggregation and abstraction [37]. As will be stated in the Problem Formulation section (Sec. IV), the experimentation in this paper is based on a fully observable MDP; i.e., agents have complete knowledge of all current states without restrictions.

B. Temporal-Difference Methods

TD techniques are contained in the sample-based learning gamut. State–action value estimates $Q^i(S_t, A_t^i)$ are used to numerically quantify the execution of certain action A_t^i in S_t joint states at the t time step by any i agent. Each intelligent agent computes, stores, updates, and uses its own estimates in the tables (tabular setting) to improve decision making:

$$Q^i(S_t, A_t^i) \leftarrow Q^i(S_t, A_t^i) + \alpha\left[\hat{G}_t^i - Q^i(S_t, A_t^i)\right] \tag{2}$$

Commonly represented in sample-based learning, the incremental update rule [Eq. (2)] uses the $\hat{G}_t^i$ return estimate as the target of the update, from which the old state–action value is subtracted, forming the TD error. Notice that α step size parameter, also known as a learning rate, moderates the influence of the error in the equation.

Inspired by the recursive form of Bellman equations, TD methods define the return as a succession of state–action values.

For a given policy π^i (agent behavior), the Bellman state–action function (i.e., expected return given state s and action a) is characterized as

$$q^i_\pi(s,a) \doteq E_{\pi^i}[G^i_t|S_t{=}\ s, A^i_t = a] = \sum_{s'}\sum_{r} p(s',r|s,a)\left[r^i + \gamma\sum_{a'}\pi^i(a'|s')q^i_\pi(s',a')\right] \quad (3)$$

Environment dynamics p, which are also seen as the probability of transitioning to the next joint states s' and receive r^i reward given s and a, are often unknown. Consequently, the p term is usually enclosed in α learning rate [Eq. (2)], which is tuned through hyperparameter analysis.

Given s', the expectation term

$$\sum_{a'}\pi^i q^i_\pi$$

in Eq. (3) bootstraps the likelihood of selecting the next action by multiplying it by its respective state–action value. Although the expected SARSA return is defined as in Eq. (3), SARSA employs the next state–action value $Q^i_{t.\pi}(S_{t+1}, A^i_{t+1})$, and Q learning[§] exploits the maximum state–action value max $Q^i(S_{t+1}, a')$. Off-policy methods such as the expected SARSA and Q learning can adopt exploratory behaviors without affecting state–action value updates, which is a powerful technique to balance exploration and exploitation. On-policy methods, instead, require specific policies like ε greedy [17] to achieve that equilibrium. The experimentation carried out in this research uses an ε-greedy policy for every agent.

C. n-Step Dynamic Tree Search

As a tree search algorithm, the proposed NSDTS incorporates forward planning mechanisms and real-world updates. Let model-based learning be decomposed in two submodules: a probabilistic behavior model, and an environment model. The first is responsible for forecasting future evader movement patterns based on current joint states S_t. The latter determines S_{t+1} joint states' transition by greedily selecting action A^i_t and assuming the best play of team agents. Following Fig. 2, after N future samples, updates are back-propagated to the root node of the tree. The estimated return for N forward steps is presented in Eq. (4):

$$\hat{G}^i_{\tau:\tau+N} = R^i_{\tau+1} + \gamma\sum_{a\neq A^i_{\tau+1}}\pi^i(a|S_{\tau+1})Q^i_{\tau+N-1}(S_{\tau+1},a) + \gamma\pi^i(A^i_{\tau+1}|S_{\tau+1})\hat{G}^i_{\tau+1:\tau+N} \quad (4)$$

Note the difference between real time t and forward fictitious time τ. According to the TD incremental update rule in Eq. (2), the Q^i table is first refreshed at the $\tau + N - 1$ time step, and then the subscript. Furthermore, for the special cases of $\tau + 1 = N$ and $\tau + 1 = T$, where $\tau = \min(T, t + [0, 1, \ldots, N - 1])$, the estimated return is computed either as an expected TD update and refers to Eq. (5)

$$\hat{G}^i_\tau{=}\ R^i_{\tau+1} + \gamma\sum_a \pi^i(a|S_{\tau+1})Q^i(S_{\tau+1},a) \quad (5)$$

or just $\hat{G}^i_\tau{=}\ R^i_T$, respectively. Equation (5) coincides with the expected SARSA return; therefore, the state–action values are softened with the π^i probability distribution. Note that Eq. (4) is the multistep version of Eq. (5).

Once the model-learning stage ends, agent i gains real-world experience through the enhanced policy π^i. The full NSDTS algorithm is disclosed step by step in Algorithm 1. Notice how future sample trajectories and direct updates modify state-action values simultaneously.

[§]Note that the expected SARSA is a generalization of Q learning because both expressions are equivalent if π^i is a greedy policy.

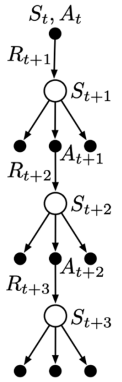


Fig. 2 Three-step tree: backup update [17].

Algorithm 1: n-step dynamic tree search algorithm for ε-greedy policy π^i

Initialize $Q^i(s,a)$ for $i \in [1,n]$ learning agents, $s \in S$ joint states, and $a \in A(s)$
Initialize net(s) probabilistic behavior model and model(s,a) environment model
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$ for policy π^i, planning steps N
Loop for each episode ($t = 0, 1, \ldots, T-1$):
  Initialize and store S_0 joint states (not terminal)
  Loop for n agents ($i = 0, 1, \ldots, n-1$):
    Loop for N planning steps ($\tau = t + [0, 1, \ldots, N-1]$):
      Predict a_τ prey action with net(S_τ)
      $A^i_\tau, R^i_{\tau+1}, S_{\tau+1} \leftarrow$ model(S_τ, a_τ) assuming best play
      Store $A^i_\tau, R^i_{\tau+1}, S_{\tau+1}$
      Break loop if $\tau + 1 = T$
    if $\tau + 1 = T$:
      $G^i_\tau \leftarrow R^i_T$
    else:
      $G^i_\tau \leftarrow R^i_{\tau+1} + \gamma\sum_a \pi^i(a|S_{\tau+1})Q^i(S_{\tau+1},a)$
    $Q^i(S_\tau, A^i_\tau) \leftarrow Q^i(S_\tau, A^i_\tau) + \alpha[G^i_\tau - Q^i(S_\tau, A^i_\tau)]$
    Loop for $k = \tau$ down to $t+1$:
      $G^i_{k-1} \leftarrow R^i_k + \gamma\sum_{a\neq A^i_k}\pi^i(a|S_k)Q^i(S_k,a) + \gamma\pi^i(A^i_k|S_k)G^i_k$
      $Q^i(S_{k-1}, A^i_{k-1}) \leftarrow Q^i(S_{k-1}, A^i_{k-1}) + \alpha[G^i_{k-1} - Q^i(S_{k-1}, A^i_{k-1})]$
    Select A^i_t following π^i
  Observe and store R_{t+1}, S_{t+1}
  Loop for n agents ($i = 0, 1, \ldots, n-1$):
    if $t + 1 < T - 1$:
      $G^i_t \leftarrow R^i_{t+1} + \gamma\sum_a \pi^i(a|S_{t+1})Q^i(S_{t+1},a)$
    else:
      $G^i_t \leftarrow R^i_{t+1}$
    $Q^i(S_t, A^i_t) \leftarrow Q^i(S_t, A^i_t) + \alpha[G^i_t - Q^i(S_t, A^i_t)]$

III. System Architecture

The architecture of the deployed system is presented in Fig. 3. Agents can access, check, or update values within their decentralized Q tables to either execute forward planning or simply select a greedy action. The MDP is fully observable; i.e., each agent detects all joint state transitions from the environment. However, actions taken by team agents are individually assumed, and hence may differ from actual behaviors. Despite the actual disagreements, these assumptions imply system robustness, leading to conservative policies. Swarming enhancements via data exchange and team agreements figure into future work.

The probabilistic behavior model consists of a neural network that predicts the evader's next action given S_t joint states. The network configuration is depicted in Fig. 3 for two different problem settings. The stochastic prey study analyzes the cooperative performance of two learning hunters against an evader with a uniform random policy. The second problem setting considers an intelligent prey that escapes

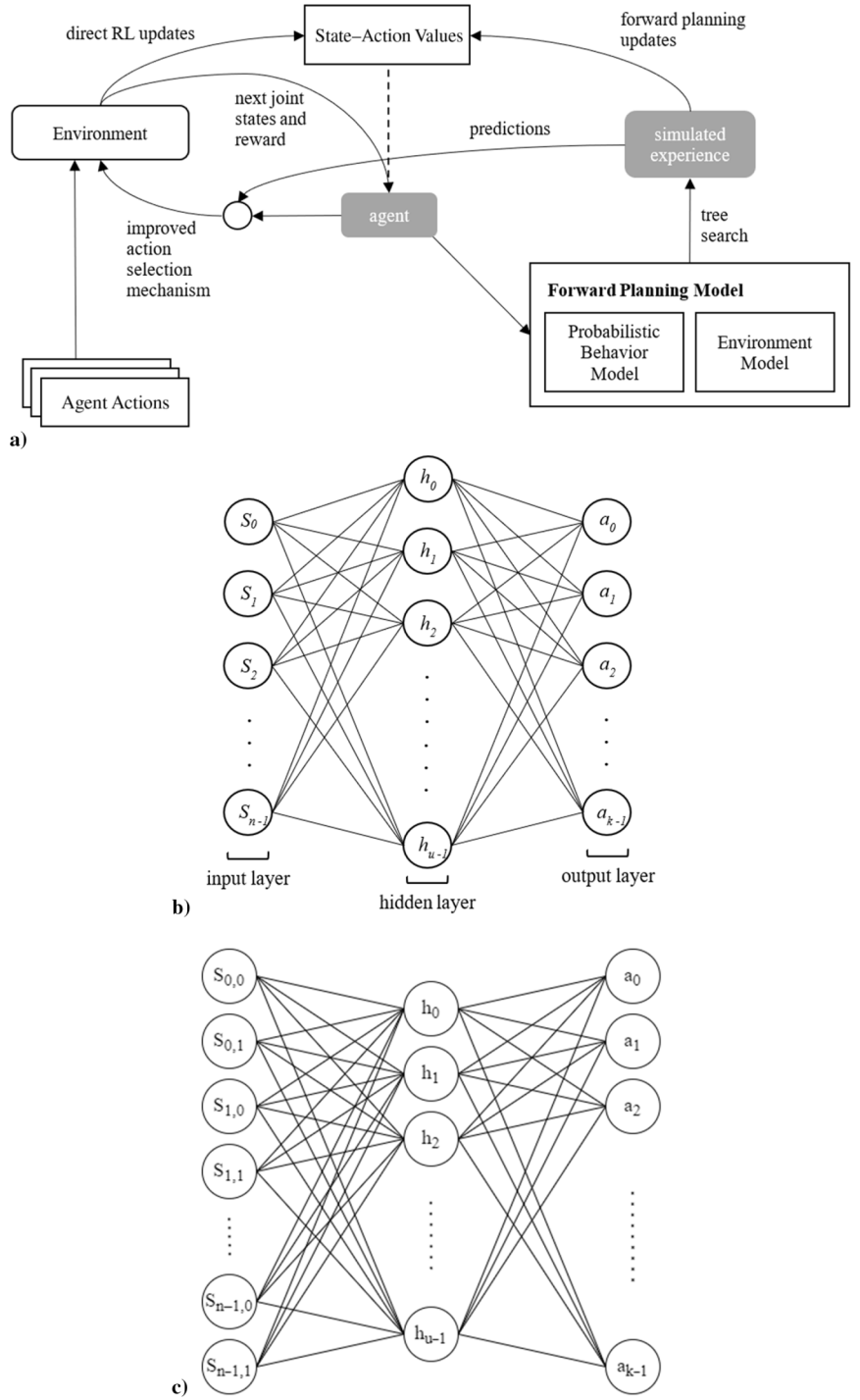


Fig. 3 NSDTS system overview from an individual agent perspective (Fig. 3a), probabilistic behavior model for stochastic prey study (Fig. 3b), and probabilistic behavior model for intelligent prey study (Fig. 3c).

from pursuers, given their global positions. Hence, to achieve accurate predictions, the latter approximator has twice as many input neurons as the joint MDP states, corresponding to the Cartesian decomposition of agent locations. In both cases, the number of hidden neurons u remains a hyperparameter, whereas the output layer is constituted by the available actions k that an agent can execute.

This can also be thought of as a classifier problem, with an assortment of MDP states as the input and several action classes as the output. Thereupon, the error of prediction is computed through a cross-entropy function. Optimization is implemented via backpropagation[¶] using ADAM [38]. This upgraded version of *stochastic gradient descent* leverages adaptive vector step sizes (ergo, greater learning rates for parameters with higher errors) and low-order moments that progressively boost weight gradients toward constant directions. Altogether, these techniques improve approximator learning processes, reaching global or local minima faster.

IV. Problem Formulation

The hunter–prey pursuit problem has been a benchmark in MARL for years. Continuous state spaces [34,36], grid domains [25,26], and obstacles [24] are some environmental traits considered by the authors. This section contains key aspects and fundamental rules that characterize the experimentation setup.

[¶]This is a technique that computes the gradient of the loss function (error) with respect to the function approximation parameters.

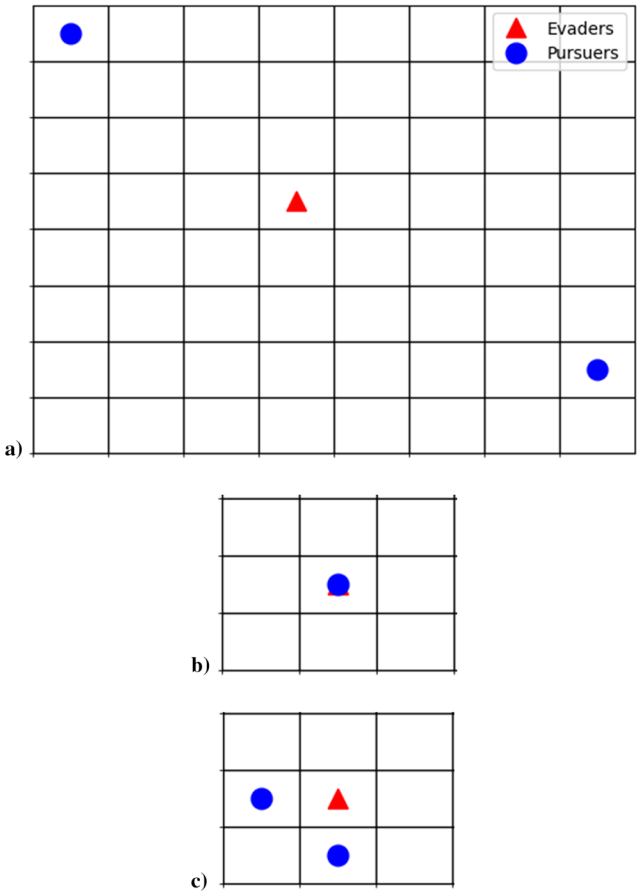


Fig. 4 Hunter–prey pursuit game environment (Fig. 4a), individual capture (Fig. 4b), and cooperative capture (Fig. 4c).

Figure 4 reveals an eight-by-eight grid world with an evader and two intelligent pursuers. The prey is placed in the center of the grid at the start of each episode, whereas hunters are randomly assigned to nonterminal states. Pursuers must capture the evader without colliding. Individual captures occur when a hunter and a prey occupy the same cell, whereas cooperative captures arise when two or more pursuers are in the evader's closest adjacent cells. To encourage cooperation, captures involving multiple team agents receive higher rewards (see Table 1).

Collisions, on the other hand, happen either when two or more hunters occupy the same cell at the same time or if a hunter on the grid's edge conducts an action toward the wall. Agents engaged in a crash are penalized. Furthermore, to encourage hunting efficiency, pursuers are given a negative compensation each time step. Each agent can move up, right, and down; or, given the occasion, they can stay on the same cell.

This paper presents and evaluates the performance of two learning hunters against a stochastic evader and an intelligent prey that, following an ε-greedy policy, moves away from hunters given their position. If the evader is cornered and no possible escape actions are available, the agent randomly selects an action.

V. Experimental Results and Discussion

The gathered results from two distinct study cases are comprehended in this section.

A. Stochastic Prey Study

The NSDTS and Q-learning algorithms are compared in the hunter–prey pursuit environment, alongside a simple evader with a uniformly random policy. Because algorithm performance is deeply dependent on the α learning rate, a hyperparameter analysis is carried out first to tune α for each method. Table 1 summarizes the setup configuration adopted for the study. The algorithm parameters are applied to all methods, except from N planning steps that only apply to NSDTS. The neural network learning rate (not to be confused with α from the RL algorithms) influences the weight gradient size toward a loss function's minimum, whereas λ is used to avoid overfitting. Adaptive vector step sizes and low-order moments of the ADAM optimizer are regulated by the mean moment update β_m and the second moment update β_v. Due to the prey's uniformly random policy, predicted action accuracy is not considered as a model performance metric; output probability distribution is assessed instead. Totals of five and one million samples have been used to train and test the neural network model, achieving a model cumulative error of 4.2945×10^{-4}. The stochastic batch size (batch size $= 1$) has been considered during training. As far as the reward function is concerned, a hybrid structure (i.e., a combination of individual and global rewards) has been carefully designed to approach the credit assignment problem, where n_c refers to the number of pursuers involved in the cooperative capture.

Table 1 Hyperparameter analysis configuration: stochastic prey study

Parameter	Value
Algorithm parameters	
Planning steps N	5
Discount factor γ	1.00
Exploring parameter ε	0.10
Model features	
Input units	3
Hidden units	8
Output units	5
Neural network learning rate	10^{-3}
Weight decay λ	10^{-3}
Mean moment update parameter β_m	9×10^{-2}
Second moment update parameter β_v	9.99×10^{-2}
Batch size	1
Training samples	5,000,000
Testing samples	1,000,000
Model cumulative error	4.2945×10^{-4}
Hybrid reward function	
Colliding with boundaries or other agents	−100
Each step	−1
Individual captures	+10
Cooperative captures	$+100 \times n_c$

The hyperparameter analysis in Fig. 5 exposes the overall Q-learning and NSDTS performances for five planning steps and different step sizes. With an optimal learning of rate $\alpha = 0.040$, the NSDTS outperforms Q learning by 800% after 25,000 episodes. Besides, the agents learn at a higher pace, achieving almost 300% improvement after policy convergence. Both algorithms develop successfully cooperative behaviors; i.e., hunters learn how to execute cooperative captures as the final averaged reward climbs to around 25, 60, and 70 for the Q learning, the two-step dynamic tree search (2SDTS), and the 11-step dynamic tree search (11SDTS), respectively (surpassing individual capture reward in Table 1). The 11-step dynamic tree search presents minor enhancements with respect to its two-step version due to the inner environment simplicity; i.e., the 2SDTS almost achieves an optimal policy. However, an increasing problem complexity may require additional planning steps to reach optimal policies.

B. Intelligent Prey Study

This subsection tests the NSDTS, Q learning, expected SARSA, and SARSA against an intelligent evader that moves opposite to the hunters. Similar to Sec. V.A, Table 2 gathers all system parameters selected for the analysis. Despite the randomness involved in the ε-greedy evader's policy, the neural network model reaches 81.4% in prediction accuracy, using a total of 25 and 5 million samples for

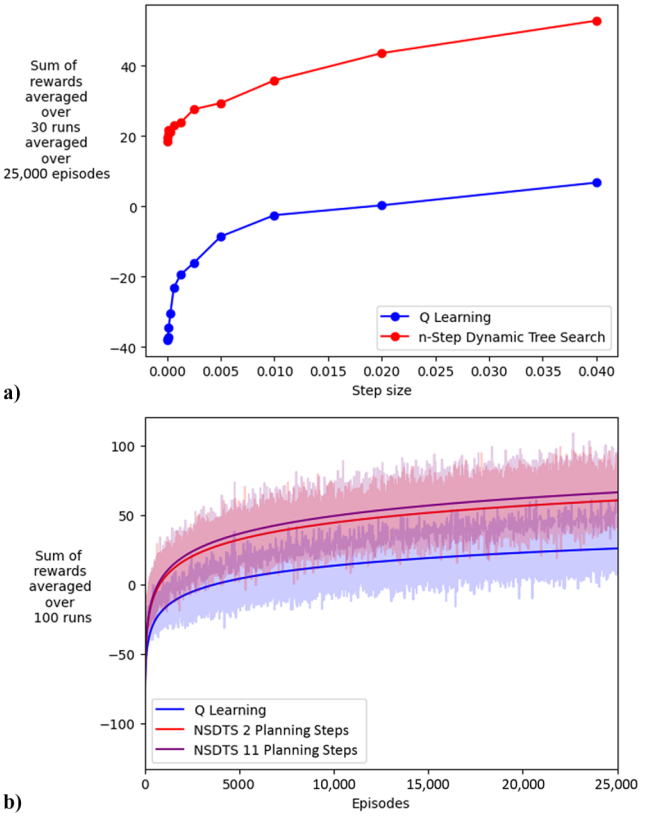


Fig. 5 *n*-step dynamic tree search performance against *Q*-learning temporal-difference method: a) step size analysis, and b) algorithm performance with optimal learning rates through averaged sum of rewards.

training and testing accordingly. To prevent overfitting, the batch size has been increased up to 500.

The results in Fig. 6 highlight the difference between the NSDTS and the rest of the TD methods. After 25,000 episodes, for a learning rate of $\alpha = 0.080$, the five-step dynamic tree search's overall performance is more than 50 times better than the rest, which represents an improvement of greater than 5000%. An analysis based on optimal α demonstrates higher learning

Table 2 Hyperparameter analysis configuration, intelligent prey study

Parameter	Value
Algorithm parameters	
Planning steps N	5
Discount factor γ	1.00
Exploring parameter ε	0.10
Model features	
Input units	6
Hidden units	8
Output units	5
Neural network learning rate	10^{-3}
Weight decay λ	10^{-3}
Mean moment update parameter β_m	9×10^{-2}
Second moment update parameter β_v	9.99×10^{-2}
Batch size	500
Training samples	25,000,000
Testing samples	5,000,000
Model accuracy	81.4%
Hybrid reward function	
Colliding with boundaries or other agents	-100
Each step	-1
Individual captures	$+10$
Cooperative captures	$+100 \times n_c$

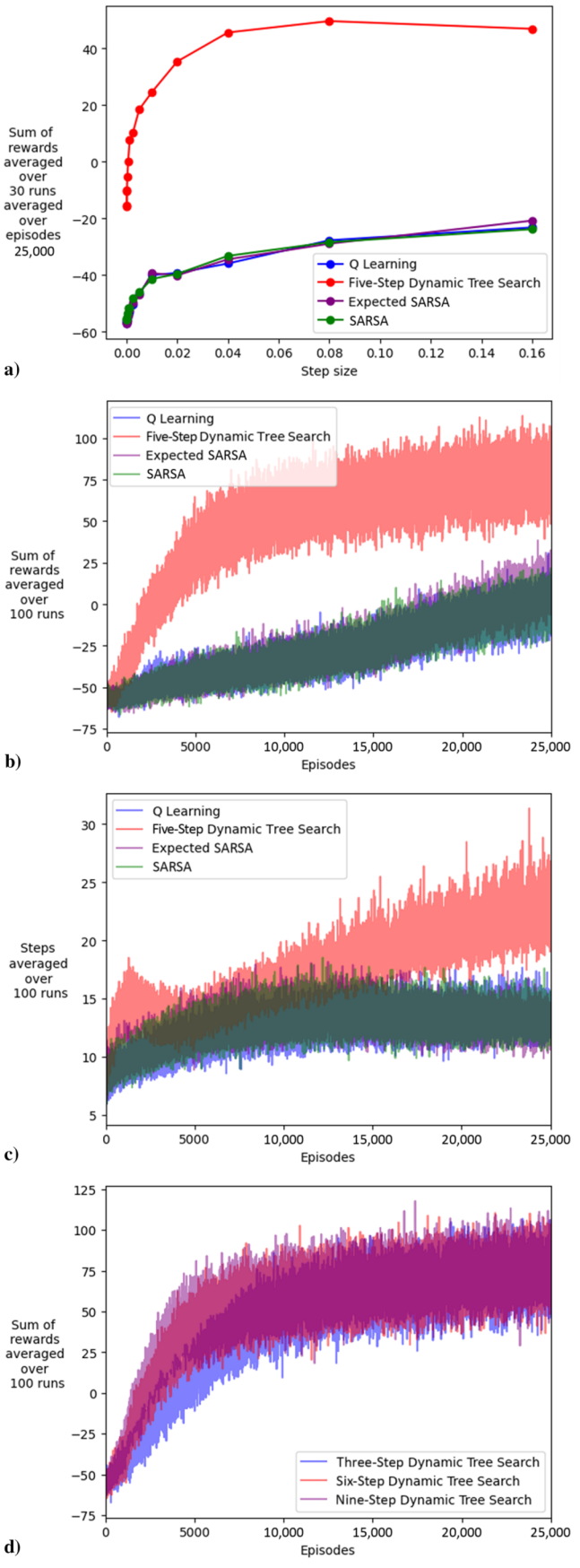
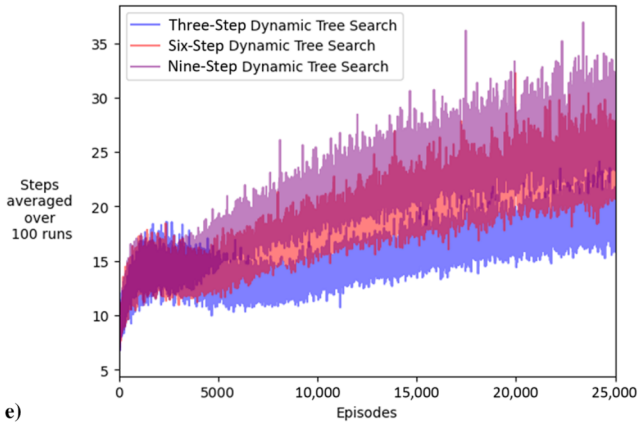


Fig. 6 Evaluating *n*-step dynamic tree search performance against *Q*-learning, expected SARSA, and SARSA temporal-difference methods: a) learning rate hyperparameter analysis, b) performance based on averaged sum of rewards, c) averaged steps with optimal step sizes, d) forward planning performance impact based on averaged sum of rewards, and e) averaged steps.



e)

Fig. 6 (Continued)

speeds on tree search agents. After 10,000 episodes, the five-step dynamic tree search nearly converged to an optimal policy, whereas conventional TD methods failed to reach suboptimal policies even after 25,000 episodes. Introducing an intelligent evader significantly increases the complexity of the problem and search space. Therefore, algorithms such as Q learning require more training to converge.

It is worth noting that the NSDTS takes more steps than any TD learning technique to reach terminal states. The combination of expected updates and forward planning allow agents to avoid collisions and choose actions conservatively, resulting in higher rewards at the expense of longer episodes. Likewise, a higher number of forward planning steps results in greater cumulative rewards. Longer tree search reasoning allows agents to learn at faster rates and avoid complex states that heavily compromise agent operation, i.e., situations where collisions can easily arise without communication.

VI. Conclusions

The n-step dynamic tree search evaluates future joint states based on an inferred model learned from real-world interactions, achieving not only an optimal performance but also high learning rates. The NSDTS presents overall improvements of 800 and 300% after policy convergence with respect to Q learning in simple stochastic scenarios. The performance gap between the multiagent tree search and traditional TD learning significantly increases in more sophisticated environments with intelligent evaders. The NSDTS manifests an improvement greater than 5000% with respect to the expected SARSA, Q learning, and SARSA: both in general performance and after policy. Thus, this paper consolidates the n-step dynamic tree search as a theoretical extension of tree search methods in the multiagent (tabular) domain.

The experimentation in this work is carried out within the hunter–prey pursuit framework, under a decentralized tabular setting, and without information exchange or agreements between team agents. Future work includes transitioning from tabular to function approximation domains to scale up the experiment, adding complexity with more agents and continuous environments (e.g., real-case applications such as drone delivery services, search and rescue (SAR) operations, surveillance, warehousing, etc.). Communication and reward optimization may be considered as well. Future work includes transitioning from tabular to function approximation domain to scale up the experiment, adding complexity with more agents and continuous environments (e.g., real-case applications such as drone delivery services, safe and rescue operations, surveillance, warehousing, etc.). Communication and reward optimization may as well be considered.

Acknowledgments

This research is sponsored by the Engineering and Physical Sciences Research Council and BAE Systems under project reference number 2454254.

References

[1] Cunningham, P., Cord, M., and Delany, S. J., "Supervised Learning," *Machine Learning Techniques for Multimedia: Case Studies on Organization and Retrieval*, edited by M. Cord, and P. Cunningham, Springer, Berlin, 2008, pp. 21–49. https://doi.org/10.1007/978-3-540-75171-7_2

[2] Geng, J., Fan, J., Wang, H., Ma, X., Li, B., and Chen, F., "High-Resolution SAR Image Classification via Deep Convolutional Autoencoders," *IEEE Geoscience and Remote Sensing Letters*, Vol. 12. No. 11, 2015, pp. 2351–2355. https://doi.org/10.1109/LGRS.2015.2478256

[3] Solan, Z., Horn, D., Ruppin, E., and Edelman, S., "Unsupervised Learning of Natural Languages," *Proceedings of the National Academy of Sciences*, Vol. 102, No. 33, 2005, pp. 11,629–11,634. https://doi.org/10.1073/pnas.0409746102

[4] Sivakumar, S., and Chandrasekar, C., "Lung Nodule Segmentation Through Unsupervised Clustering Models," *Procedia Engineering*, Vol. 38, Jan. 2012, pp. 3064–3073. https://doi.org/10.1016/j.proeng.2012.06.357

[5] Panait, L., and Luke, S., "Cooperative Multi-Agent Learning: The State of the Art," *Autonomous Agents and Multi-Agent Systems*, Vol. 11, No. 3, 2005, pp. 387–434. https://doi.org/10.1007/s10458-005-2631-2

[6] Haynes, T., Wainwright, R., Sen, S., and Schoenefeld, D., "Strongly Typed Genetic Programming in Evolving Cooperation Strategies," *Proceedings of the 6th International Conference on Genetic Algorithm*, Morgan Kaufmann, San Francisco, CA, 1995, pp. 271–278.

[7] Potter, M. A., and De Jong, K. A., "A Cooperative Coevolutionary Approach to Function Optimization," *Parallel Problem Solving from Nature*, edited by Y. Davidor, H.-P. Schwefel, and R. Männer, Springer, Berlin, 1994, pp. 249–257. https://doi.org/10.1007/3-540-58484-6_269

[8] Ficici, S. G., and Pollack, J. B., "A Game-Theoretic Approach to the Simple Coevolutionary Algorithm," *Parallel Problem Solving from Nature*, edited by M. Schoenauer, K. Deb, G. Rudolph, X. Yao, X., E. Lutton, J. J. Merelo, and H.-P. Schwefel, Springer, Berlin, 2000, pp. 467–476. https://doi.org/10.1007/3-540-45356-3_46

[9] Miconi, T., "When Evolving Populations is Better than Coevolving Individuals: The Blind Mice Problem," *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, San Francisco, CA, 2003.

[10] Claus, C., and Boutilier, C., "The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems," *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence (AAAI '98/IAAI '98)*, Assoc. for the Advancement of Artificial Intelligence (AAAI), Palo Alto, CA, 1998, pp. 746–752. https://doi.org/10.5555/295240.295800

[11] Tesauro, G., "Extending Q-Learning to General Adaptive Multi-Agent Systems," *Proceedings of the 16th International Conference on Neural Information Processing Systems (NIPS'03)*, MIT Press, Cambridge, MA, 2003, pp. 871–878. https://doi.org/10.5555/2981345.2981454

[12] Watkins, C. J. C. H., and Dayan, P., "Technical Note: Q-Learning," *Machine Learning*, Vol. 8, No. 3, 1992, pp. 279–292. https://doi.org/10.1023/A:1022676722315

[13] Sałustowicz, R. P., Wiering, M. A., and Schmidhuber, J., "Learning Team Strategies: Soccer Case Studies," *Machine Learning*, Vol. 33, No. 2, 1998, pp. 263–282. https://doi.org/10.1023/A:1007570708568

[14] Al-Dabooni, S., and Wunsch, D. C., "Online Model-Free n-Step HDP with Stability Analysis," *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 31, No. 4, 2020, pp. 1255–1269. https://doi.org/10.1109/TNNLS.2019.2919614

[15] De Asis, K., Hernandez-Garcia, J., Holland, G., and Sutton, R., "Multi-Step Reinforcement Learning: A Unifying Algorithm," *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32, , Assoc. for the Advancement of Artificial Intelligence (AAAI), Palo Alto, CA, 2018.

[16] van Seijen, H., van Hasselt, H., Whiteson, S., and Wiering, M., "A Theoretical and Empirical Analysis of Expected Sarsa," *2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, IEEE, New York, 2009, pp. 177–184. https://doi.org/10.1109/ADPRL.2009.4927542

[17] Sutton, R. S., and Barto, A. G., *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 2018, pp. 185–188, 152, 190.

[18] Foerster, J., Nardelli, N., Farquhar, G., Afouras, T., Torr, P. H. S., Kohli, P., and Whiteson, S., "Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning," *Proceedings of the 34th International Conference on Machine Learning*, edited by D. Precup, and Y. W. Teh, Proceedings of Machine Learning Research, Cambridge, MA, 2017, pp. 1146–1155.

[19] Zhang, S., and Sutton, R. S., "A Deeper Look at Experience Replay," Preprint, submitted 4 Dec. 2017, https://arxiv.org/abs/1712.01275.

[20] Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., Sepassi, R., Tucker, G., and Michalewski, H., "Model-Based Reinforcement Learning for Atari," Preprint, submitted 1 March 2019, https://arxiv.org/abs/1903.00374.

[21] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D., "Mastering the Game of Go Without Human Knowledge," *Nature*, Vol. 550, No. 7676, 2017, pp. 354–359. https://doi.org/10.1038/nature24270

[22] Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., and Silver, D., "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model," *Nature*, Vol. 588, No. 7839, 2020, pp. 604–609. https://doi.org/10.1038/s41586-020-03051-4

[23] Moradi, M., "A Centralized Reinforcement Learning Method for Multi-Agent Job Scheduling in Grid," *2016 6th International Conference on Computer and Knowledge Engineering, ICCKE 2016*. IEEE, New York, 2016, pp. 171–176. https://doi.org/10.1109/ICCKE.2016.7802135

[24] Yu, C., Dong, Y., Li, Y., and Chen, Y., "Distributed Multi-Agent Deep Reinforcement Learning for Cooperative Multi-Robot Pursuit," *Journal of Engineering*, Vol. 2020, No. 13, 2020, pp. 499–504. https://doi.org/10.1049/joe.2019.1200

[25] Ho, J., and Wang, C.-M., "Explainable and Adaptable Augmentation in Knowledge Attention Network for Multi-Agent Deep Reinforcement Learning Systems," *2020 IEEE Third International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, IEEE, New York, 2020, pp. 157–161. https://doi.org/10.1109/AIKE48582.2020.00031

[26] Tan, M., "Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents," *Readings in Agents*, Morgan Kaufmann, San Francisco, CA, 1997, pp. 487–494. https://doi.org/10.5555/284680.284934

[27] Corne, D., Reynolds, A., and Bonabeau, E., "Swarm Intelligence," *Handbook of Natural Computing*, Springer Berlin, Baden-Württemberg, Germany, 2012, pp. 1599–1623.

[28] Longa, M. E., Tsourdos, A., and Inalhan, G., "Human–Machine Network Through Bio-Inspired Decentralized Swarm Intelligence and Heterogeneous Teaming in SAR Operations," *Journal of Intelligent and Robotic Systems*, Vol. 105, No. 4, 2022, Paper 88. https://doi.org/10.1007/s10846-022-01690-5

[29] Espinós Longa, M., Inalhan, G., and Tsourdos, A., "Swarm Intelligence in Cooperative Environments: Introducing the n-Step Dynamic Tree Search Algorithm," *AIAA SciTech 2022 Forum*, AIAA Paper 2022-1839, 2022, pp. 1–13. https://doi.org/10.2514/6.2022-1839

[30] Longa, M. E., Tsourdos, A., and Inalhan, G., "Swarm Intelligence in Cooperative Environments: n-Step Dynamic Tree Search Algorithm Extended Analysis," *2022 American Control Conference (ACC)*, IEEE, New York, 2022, pp. 761–766. https://doi.org/10.23919/ACC53348.2022.9867171

[31] Abed-alguni, B. H., Chalup, S. K., Henskens, F. A., and Paul, D. J., "A Multi-Agent Cooperative Reinforcement Learning Model Using a Hierarchy of Consultants, Tutors and Workers," *Vietnam Journal of Computer Science*, Vol. 2, No. 4, 2015, pp. 213–226. https://doi.org/10.1007/s40595-015-0045-x

[32] Duman, E., Kaya, M., and Akin, E., "A Multi-Agent Fuzzy-Reinforcement Learning Method for Continuous Domains," *Multi-Agent Systems and Applications IV*, Vol. 3690, Lecture Notes in Computer Science Series (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Springer, New York, 2005, pp. 306–315. https://doi.org/10.1007/11559221_31

[33] Huang, J., Yang, B., and Liu, D.-Y., "A Distributed Q-Learning Algorithm for Multi-Agent Team Coordination," *Machine Learning and Cybernetics*, IEEE, New York, 2005, pp. 108–113. https://doi.org/10.1109/ICMLC.2005.1526928

[34] Ishiwaka, Y., Sato, T., and Kakazu, Y., "An Approach to the Pursuit Problem on a Heterogeneous Multiagent System Using Reinforcement Learning," *Robotics and Autonomous Systems*, Vol. 43, No. 4, 2003, pp. 245–256. https://doi.org/10.1016/S0921-8890(03)00040-X

[35] Kuremoto, T., Tsurusaki, T., Kobayashi, K., Mabu, S., and Obayashi, M., "An Improved Reinforcement Learning System Using Affective Factors," *Robotics*, Vol. 2, No. 3, 2013, pp. 149–164. https://doi.org/10.3390/robotics2030149

[36] Wang, Y., Dong, L., and Sun, C., "Cooperative Control for Multi-Player Pursuit-Evasion Games with Reinforcement Learning," *Neurocomputing*, Vol. 412, Oct. 2020, pp. 101–114. https://doi.org/10.1016/j.neucom.2020.06.031

[37] Daoui, C., Abbad, M., and Tkiouat, M., "Exact Decomposition Approaches for Markov Decision Processes: A Survey," *Advances in Operations Research*, Vol. 2010, July 2010, Paper 659432. https://doi.org/10.1155/2010/659432

[38] Kingma, D. P., and Ba, J., "Adam: A Method for Stochastic Optimization," *3rd International Conference on Learning Representations, ICLR 2015—Conference Track Proceedings*, International Conference on Learning Representations (ICLR), Ithaca, NY, 2014, pp. 1–15.

H. Choi
Associate Editor

2023-05-03

Swarm intelligence in cooperative environments: n-step dynamic tree search algorithm overview

Espinós Longa, Marc

AIAA

Espinós Longa M, Tsourdos A, Inalhan G. (2023) Swarm intelligence in cooperative environments: n-step dynamic tree search algorithm overview. Journal of Aerospace Information Systems, Volume 20, Issue 7, July 2023, pp. 418-425

https://doi.org/10.2514/1.I011086