In: 2022 American Control Conference (ACC), Atlanta, 8-10 June 2022
DOI: 10.23919/ACC53348.2022.9867171

Swarm Intelligence in Cooperative Environments: N-Step Dynamic Tree Search Algorithm Extended Analysis

Marc Espinós Longa, Antonios Tsourdos, and Gokhan Inalhan
Cranfield University, Bedfordshire, MK43 0AL, United Kingdom

Abstract— **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, e.g., Go and chess strategic board games. This paper pretends to extend tree search algorithms to the multi-agent 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 state-of-the-art algorithms such as Q-Learning and SARSA. Future state transitions and rewards are predicted with a model built and learned from real interactions between agents and the environment. As an extension of previous work, this paper analyses the developed algorithm in the Hunter-Pursuit cooperative game against intelligent evaders. The N-Step Dynamic Tree Search aims to adapt the most successful single-agent learning methods to the multi-agent boundaries and demonstrates to be a remarkable advance compared to conventional temporal-difference techniques.**

I. Introduction

Machine Learning (ML) field has been expanding over the last decades because of its wide range of application. Typically, a learning process can be classified in three main groups: supervised [1][2], unsupervised [3][4] and reward-based learning methods [5]. Addressing single-agent and multi-agent domains often entail 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), an efficient technique for small state spaces. Darwinian models of evolution are used to refine populations of candidate behaviors in evolutionary computation. *Genetic Programming* (GP) [6] and *Coevolutionary Algorithms* (CEAs) [7][8][9] are solid contenders within this field. On the other side, *Reinforcement Learning* (RL) uses value functions under a *Markov Decision Process* (MDP) framework, 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][11][12][13] to n-step algorithms that wait a variable or fixed number of MDP transitions before recomputing 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. In contrast, Monte Carlo methods, which are the most extreme form of n-step algorithms, delay updates until the end of the episode, removing any bias and significantly increasing the variance of the optimization process. As Fig. 1 [16] states, the type of update constitutes the wide dimension of the sample-based learning map. Expectations [17] contemplate all potential trajectories, weighting value estimates at a higher computational cost.

Planning is another powerful tool to accelerate learning by alternating direct RL updates from real-world interactions with simulated trajectories, 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 raises tree search single-agent methods to the next level. Forward planning through an existing model is used to make predictions and optimize action selection mechanisms. *Monte Carlo Tree*

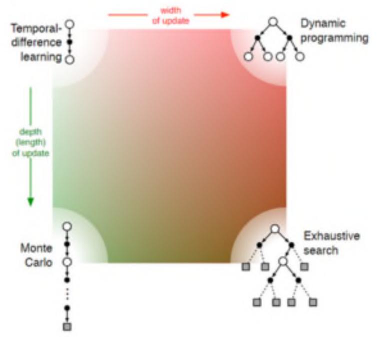


Fig. 1 Simplified map of sample-based learning methods

*Research supported by Engineering and Physical Sciences Research Council (EPSRC) and BAE Systems under the project reference no. 2454254.

Marc Espinós Longa is a PhD Researcher in the School of Aerospace, Transport & Manufacturing at Cranfield University, Bedfordshire, MK43 0AL, United Kingdom (e-mail: marc.espinos-longa@cranfield.ac.uk).

Antonios Tsourdos is an AIAA Senior Member, Head of Center and Director of Research in the School of Aerospace, Transport & Manufacturing at Cranfield University, Bedfordshire, MK43 0AL, United Kingdom (e-mail: a.tsourdos@cranfield.ac.uk)

Gokhan Inalhan is AIAA Associate Fellow, BAE Systems Chair and Deputy Head of Center in the School of Aerospace, Transport & Manufacturing at Cranfield University, Bedfordshire, MK43 0AL, United Kingdom (e-mail: inalhan@cranfield.ac.uk)

© 2022 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works

Search (MCTS) [16] becomes a standard in this collective, and recently inspires many powerful algorithms such *MuZero* developed by DeepMind, validated and tested with superhuman performance across a large variety of Atari games [22].

Transitioning from single-agent to multi-agent framework is no trivial task. Even from a *team learning* perspective [5] where agents are treated as a master individual and single-agent techniques apply, big issues emerge related to enormous state spaces (curse of dimensionality problem) that grow exponentially to the number of agents. Furthermore, like any other centralized system, team learning requests full swarm connectivity and is reliant on a critical leading node. On the other hand, c*oncurrent learning* [5] utilizes a distributed network of agents to handle multiple learning processes simultaneously [23][24]. Thus, co-adaption through reinforcements is required to achieve optimal cooperation. The credit assignment problem involves allocating efficient incentives to individuals and remains a difficult endeavor in the *Multi-Agent Reinforcement Learning* (MARL) problem. Similarly, communication is crucial to enhance the learning process. While some techniques use direct communication to share TD-updates, policies or even full episodes [25], others opt for indirect bio-inspired mechanisms [26][27].

This study aims to apply the most successful single-agent RL algorithms to the multi-agent domain as a decentralized system. Inspired by MCTS and n-Step Tree Backup updates [16], the *N-Step Dynamic Tree Search* (NSDTS) algorithm combines forward planning with direct RL, achieving a major performance breakthrough over traditional TD techniques like Q-Learning. Action selection mechanisms are improved by using a neural network model, built from agent experience, to perform forward tree search. To guarantee system robustness, communication has been disregarded and left for future work; thence, individuals assume *best play* of agents to carry out future predictions. Agents are completely independent, have their own *q tables* (tabular setting) and learn entirely from their individual environment readings. Expected updates are utilized to weight state-action value estimates, and agents are reinforced with hybrid rewards (individual and global incentives).

As an extension of previous work [28], 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 a number of learning agents cooperating to capture one or more evaders, has been a reference to MARL nearly since its inception [25][29][30][31][32][33][34]. This work conceives a grid world with two learning hunters and an intelligent prey that tries to escape from chasers given their position.

The remainder of the paper is structured as follows: theoretical foundation in MARL is provided in Section 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, considering environment features, game rules, and reward function. Results and discussion of hyperparameter analysis and algorithm performance experimentation are presented in Section V. Concluding marks and future work suggestions are given in Section VI.

II. Theoretical Background

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, 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 multi-agent frames. In the MARL configuration, for a given time *t* and a set of *n* agents in $S_t = (S_t^0, S_t^1, \dots, S_t^{n-2}, S_t^{n-1})$ joint states, after executing $A_t = (A_t^0, A_t^1, \dots, A_t^{n-2}, A_t^{n-1})$ actions through some action selection mechanism, the environment transitions all agents to the next MDP state $S_{t+1} = (S_t^0, S_{t+1}^1, \dots, S_{t+1}^{n-2}, S_{t+1}^{n-1})$ 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 $< S_0, A_0, R_1, S_1, A_1, \dots, R_{T-1}, S_{T-1} >$ that at time t = T reach a terminal joint state.

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

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \, ; \qquad (1)$$

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

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* (POMDPs) [25]. *Factored* MDPs (FMDPS) promote parallel processing via state aggregation and abstraction [35]. As states in Section IV Problem Formulation, the experimentation in this paper is based on a fully-observable MDP, i.e., agents have complete knowledge 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 t time step by any i agent. Each intelligent agent computes, stores, updates and uses its own estimates in tables (tabular setting) to improve decision making.

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

Commonly represented in sample-based learning, the incremental update rule (2) uses $\widehat{G}_t^i$ 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 learning rate, moderates the influence of the error in the equation.

Each TD method uses a specific target for its state-action update rule. For a given policy π^i, Expected SARSA return is $\widehat{G}_t^i = R_{t+1}^i + \gamma \sum_{a'} \pi^i(a'|s') Q_{t,\pi}^i(s', a')$, being R_{t+1}^i the next reward, and the summatory term the expectation of next state-action values. Differently, SARSA employs the next visited state-action value $Q_{t,\pi}^i(S_{t+1}, A_{t+1}^i)$, and Q-Learning[1] exploits the next maximum state-action value $\max Q^i(S_{t+1}, a')$. Off-policy methods such as Expected SARSA and Q-Learning can adopt exploratory behaviors without affecting state-action value updates, a powerful technique to balance exploration and exploitation. On-policy methods instead, require specific policies like ε-greedy [16] to achieve that equilibrium.

C. N-Step Dynamic Tree Search

As a tree search algorithm, NSDTS incorporates forward planning mechanisms and direct 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_t^i assuming best play of team agents. Following Fig. 2 [16], after N future samples, updates are backpropagated until the root node of the tree. The estimated return for N forward steps is presented in (3).

$$\widehat{G}_{\tau:\tau+N}^i = R_{\tau+1}^i + \gamma \sum_{a \neq A_{\tau+1}^i} \pi^i(a|S_{\tau+1}) Q_{\tau+N-1}^i(S_{\tau+1}, a) + \gamma \pi^i\left(A_{\tau+1}^i \middle| S_{\tau+1}\right) \widehat{G}_{\tau+1:\tau+N}^i \tag{3}$$

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

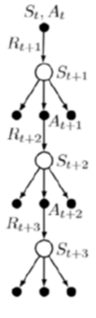


Fig. 2 3-Step Tree-Backup update

[1] Note that Expected SARSA is a generalization of Q-Learning since both expressions are equivalent if π^i is a greedy policy.

Table 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 $\pi^i$, planning steps N
Loop for each episode ($t = 0, 1, \dots, T - 1$):
  Initialize and store $S_0$ joint states (not terminal)
  Loop for n agents ($i = 0, 1, \dots, n - 1$):
    Loop for N planning steps ($\tau = t + [0, 1, \dots, N - 1]$):
      Predict $a_\tau$ prey action with net($S_\tau$)
      $A_\tau^i, R_{\tau+1}^i, S_{\tau+1} \leftarrow$ Model($S_\tau, a_\tau$) assuming best play
      Store $A_\tau^i, R_{\tau+1}^i, S_{\tau+1}$
      Break loop if $\tau + 1 = T$
    if $\tau + 1 = T$:
      $G_\tau^i \leftarrow R_T^i$
    else:
      $G_\tau^i \leftarrow R_{\tau+1}^i + \gamma \sum_a \pi^i(a|S_{\tau+1}) Q^i(S_{\tau+1}, a)$
    $Q^i(S_\tau, A_\tau^i) \leftarrow Q^i(S_\tau, A_\tau^i) + \alpha[G_\tau^i - Q^i(S_\tau, A_\tau^i)]$
    Loop for $k = \tau$ down to $t + 1$:
      $G_{k-1}^i \leftarrow R_k^i + \gamma \sum_{a \neq A_k^i} \pi^i(a|S_k) Q^i(S_k, a) + \gamma \pi^i(A_k^i|S_k) G_k^i$
      $Q^i(S_{k-1}, A_{k-1}^i) \leftarrow Q^i(S_{k-1}, A_{k-1}^i) + \alpha[G_{k-1}^i - Q^i(S_{k-1}, A_{k-1}^i)]$
    Select $A_t^i$ following $\pi^i$
  Observe and store $R_{t+1}, S_{t+1}$
  Loop for n agents ($i = 0, 1, \dots, n - 1$):
    if $t + 1 < T - 1$:
      $G_t^i \leftarrow R_{t+1}^i + \gamma \sum_a \pi^i(a|S_{t+1}) Q^i(S_{t+1}, a)$
    else:
      $G_t^i \leftarrow R_{t+1}^i$
        $Q^i(S_t, A_t^i) \leftarrow Q^i(S_t, A_t^i) + \alpha[G_t^i - Q^i(S_t, A_t^i)]$
```

$$\widehat{G}_\tau^i = R_{\tau+1}^i + \gamma \sum_a \pi^i(a|S_{\tau+1}) Q^i(S_{\tau+1}, a), \tag{4}$$

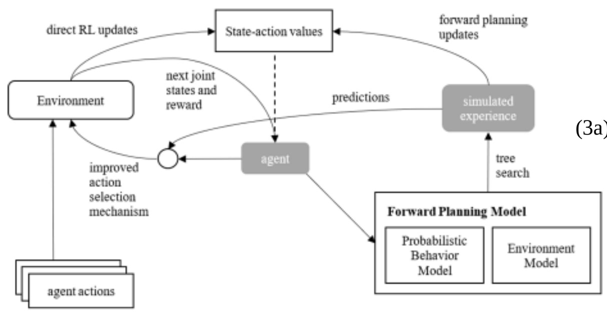
or just $\widehat{G}_\tau^i = R_T^i$ respectively. Equation (4) coincides with Expected SARSA return; therefore, state-action values are counterbalanced with π^i probability distribution. Note that (3) is the multi-step version of (4).

Once model-learning stage ends, agent i gains experience through the enhanced policy π^i. The full NSDTS algorithm is disclosed step by step in Table 1. The N-Step Dynamic Tree Search improves agent policies and learning rates by integrating future sample trajectories and direct updates.

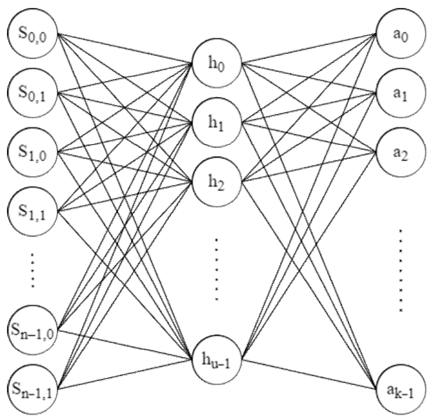
III. System Architecture

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 apparent imbalances, these assumptions imply system robustness (no communication between agents).

The probabilistic behavior model consists of a neural network that predicts the evader's next action given S_t joint



(3a)



(3b)

Fig. 3 NSDTS system overview from an individual agent perspective (3a), probabilistic behavior model (3b)

states. Network configuration is depicted in Fig. 3. The approximator has twice as many input neurons as joint MDP states, corresponding to the cartesian decomposition of the environment. The number of hidden neurons u remains a hyperparameter, while the output layer is constituted by the available actions k that an agent can execute.

As aforementioned, this model forecasts next prey actions based on a resulting probability distribution. This can also be thought as a classifier problem, with an assortment of MDP states as the input, and several action classes the output. Thereupon, the error of prediction is computed through a cross-entropy function. Optimization is implemented via backpropagation[2] using ADAM [36]. This upgraded version of *Stochastic Gradient Descent* (SGD) leverages adaptive vector step sizes, ergo, greater learning rates for parameters with higher errors, and low-order moments that progressively boost weight gradients towards constant directions. Altogether, these techniques improve approximator learning processes reaching global or local minimums faster.

IV. Problem Formulation

The hunter-prey pursuit problem has been a benchmark in MARL for years. Continuous state spaces [32][34], grid domains [24][25] and obstacles [23] are some environmental traits considered by authors. This section contains key aspects and fundamental rules that characterize the experimentation setup.

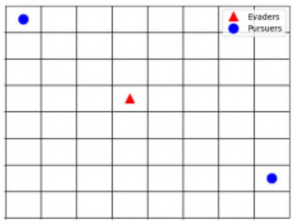


Fig. 4 Hunter-prey pursuit game environment

Fig. 4 reveals an 8 by 8 grid world with an evader and two intelligent pursuers, totaling 1,310,729 MDP states in the given representation. The prey is placed in the center of the grid at the start of each episode, whereas hunters are randomly assigned to non-terminal states. Pursuers must capture the evader without colliding. Individual captures occur when a hunter and a prey occupy the same cell, while 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.

Collisions, on the other hand, happen either when two or more hunters occupy the same cell, or if a hunter on the grid's edge conducts an action towards the wall. Anyhow, 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, down, and given the occasion, stay on the same cell.

Following part of the future work presented in [28], this paper presents and evaluates the performance of hunters against 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

This section compares NSDTS, Q-Learning, Expected SARSA and SARSA in the presented setup. Since algorithm performance is deeply dependent on α learning rate, a hyperparameter analysis is carried out first to tune α for each method. Table 2 summarizes the setup configuration adopted for the study. Note that all algorithms use an ε-greedy policy. A hybrid reward 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.

Hyperparameter analysis results in Fig. 5 already remark the difference between NSDTS and the rest of TD methods. After 25,000 episodes, for a learning rate $\alpha = 0.080$, 5-Step Dynamic Tree Search overall performance is more than 50 times better than the rest, which represents an improvement greater than 5,000%. Additionally, performance based on the averaged sum of rewards demonstrates that tree search achieves greater learning rates. After 10,000 episodes, 5SDTS nearly converged to the optimal policy, while conventional TD methods fail to reach a suboptimal policy in 25,000 episodes. Introducing an intelligent evader significantly

[2] Technique that computes the gradient of the loss function (error) with respect to the function approximation parameters.

Table 2 Hyperparameter analysis configuration

Algorithm Parameters	
Discount factor (γ)	1.00
Exploring parameter (ε)	0.10
Model and Optimizer 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)	$9 \cdot 10^{-2}$
Second moment update parameter (β_v)	$9.99 \cdot 10^{-2}$
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 x n_c

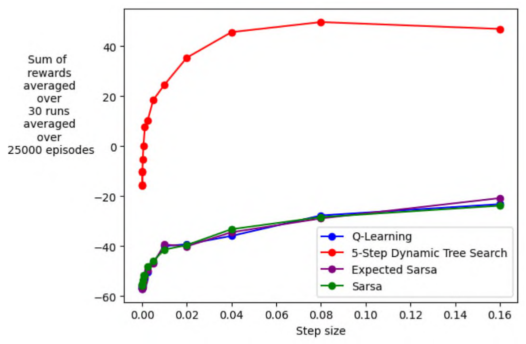
increases the complexity of the problem and search space. Therefore, algorithms such as Q-Learning require more training to converge.

It is worth noting that at the end of the performance analysis, NSDTS develops successfully cooperative policies due to the remarkable high reward. Nevertheless, it takes more steps to reach a terminal state. The combination of expected updates and forward planning allow agents to avoid collisions and choose actions conservatively, resulting in higher rewards at expense of longer episodes. Likewise, higher number of forward planning steps result in greater cumulative rewards. Longer tree search reasoning allows agents to learn at faster rates and avoid complex states that heavily compromise agent performance, i.e., situations where collisions can easily arise without communication. For the given environment, few forward steps are required to reach optimal performance after convergence. However, an increasing problem complexity may require additional planning steps to reach optimal policies.

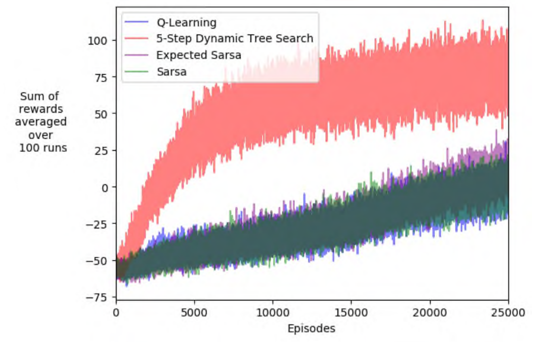
VI. Conclusion and Future Work

N-Step Dynamic Tree Search evaluates future joint states based on an inferred model learned from real-world interactions, achieving not only a striking performance, but also incredibly high learning rates. NSDTS presents an improvement superior to 5,000% with respect to Expected SARSA, Q-Learning and SARSA, both after policy convergence and as overall performance. Thus, this paper consolidates NSDTS as a significant advance in the multi-agent domain compared to conventional MARL techniques.

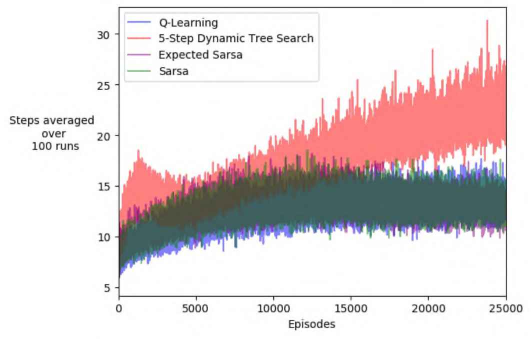
The experimentation in this work is carried out within the hunter-prey pursuit framework, under a decentralized tabular setting and without information exchange between team agents. Future work includes transitioning from tabular to function approximation domain to scale up the experiment, adding complexity to the environment with more agents (evaders and pursuers) and bigger continuous worlds with obstacles. Communication and reward optimization may as well be considered.



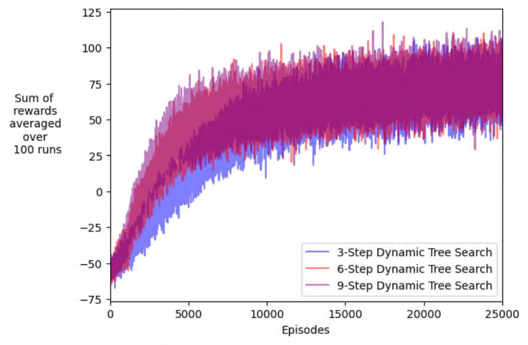
(5a)



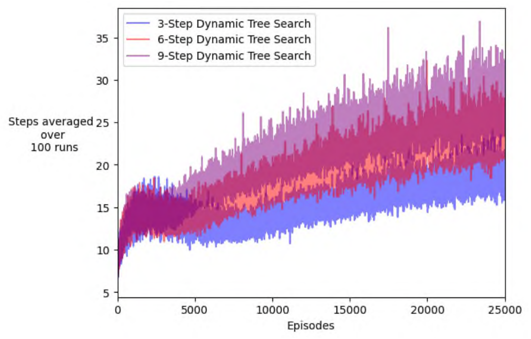
(5b)



(5c)



(5d)



(5e)

Fig. 5 Learning rate hyperparameter analysis (5a), and performance based on averaged sum of rewards (5b) and averaged steps (5c). Likewise, impact of forward planning steps displayed in (5d) and (5e).

REFERENCES

[1] Cunningham, P., Cord, M. and Delany, S. J. (2008) 'Supervised Learning', in Cord, M. and Cunningham, P. (eds) *Machine Learning Techniques for Multimedia: Case Studies on Organization and Retrieval*. Berlin, Heidelberg, pp. 21–49. doi: 10.1007/978-3-540-75171-7_2.

[2] Geng, J., Fan, J., Wang, H., Ma, X., Li, B. and Chen, F. (2015) 'High-Resolution SAR Image Classification via Deep Convolutional Autoencoders', *IEEE Geoscience and Remote Sensing Letters*. IEEE, 12(11), pp. 2351–2355. doi: 10.1109/LGRS.2015.2478256.

[3] Solan, Z., Horn, D., Ruppin, E. and Edelman, S. (2005) 'Unsupervised learning of natural languages', *Proceedings of the National Academy of Sciences*, 102(33), pp. 11629–11634. doi: 10.1073/pnas.0409746102.

[4] Sivakumar, S. and Chandrasekar, C. (2012) 'Lung Nodule Segmentation through Unsupervised Clustering Models', *Procedia Engineering*, 38, pp. 3064–3073. doi: 10.1016/j.proeng.2012.06.357.

[5] Panait, L. and Luke, S. (2005) 'Cooperative Multi-Agent Learning: The State of the Art', *Autonomous Agents and Multi-Agent Systems*, 11(3), pp. 387–434. doi: 10.1007/s10458-005-2631-2.

[6] Haynes, T., Wainwright, R., Sen, S. and Schoenefeld, D. (1995) 'Strongly Typed Genetic Programming in Evolving Cooperation Strategies.', in *Proceedings of the 6th International Conference on Genetic Algorithm*, pp. 271–278.

[7] Potter, M. A. and De Jong, K. A. (1994) 'A cooperative coevolutionary approach to function optimization', in Davidor, Y., Schwefel, H.-P., and Männer, R. (eds) *Parallel Problem Solving from Nature*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 249–257. doi: 10.1007/3-540-58484-6_269.

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

[9] Miconi, T. (2003) 'When Evolving Populations is Better than Coevolving Individuals: The Blind Mice Problem', in *Proceedings of the 18th International Joint Conference on Artificial Intelligence*.

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

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

[12] Watkins, C. J. C. H. and Dayan, P. (1992) 'Technical Note: Q-Learning', *Machine Learning*, 8(3), pp. 279–292. doi: 10.1023/A:1022676722315.

[13] Sałustowicz, R. P., Wiering, M. A. and Schmidhuber, J. (1998) 'Learning Team Strategies: Soccer Case Studies', *Machine Learning*, 33(2), pp. 263–282. doi: 10.1023/A:1007570708568.

[14] Al-Dabooni, S. and Wunsch, D. C. (2020) 'Online Model-Free n-Step HDP With Stability Analysis', *IEEE Transactions on Neural Networks and Learning Systems*, 31(4), pp. 1255–1269. doi: 10.1109/TNNLS.2019.2919614.

[15] De Asis, K., Hernandez-Garcia, J., Holland, G. and Sutton, R. (2018) 'Multi-Step Reinforcement Learning: A Unifying Algorithm', *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1 SE-AAAI Technical Track: Machine Learning). Available at: https://ojs.aaai.org/index.php/AAAI/article/view/11631.

[16] Sutton, R. S. and Barto, A. G. (2018) *Reinforcement Learning : An Introduction*.

[17] Seijen, H. van, Hasselt, H. van, Whiteson, S. and Wiering, M. (2009) 'A theoretical and empirical analysis of Expected Sarsa', in *2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pp. 177–184. doi: 10.1109/ADPRL.2009.4927542.

[18] Foerster, J., Nardelli, N., Farquhar, G., Afouras, T., Torr, P. H. S., Kohli, P. and Whiteson, S. (2017) 'Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning', in Precup, D. and Teh, Y. W. (eds) *Proceedings of the 34th International Conference on Machine Learning*. PMLR (Proceedings of Machine Learning Research), pp. 1146–1155. Available at: http://proceedings.mlr.press/v70/foerster17b.html.

[19] Zhang, S. and Sutton, R. S. (2017) 'A Deeper Look at Experience Replay', *CoRR*, abs/1712.0. Available at: http://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. (2019) 'Model-Based Reinforcement Learning for Atari', *CoRR*, abs/1903.0. Available at: http://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. (2017) 'Mastering the game of Go without human knowledge', *Nature*, 550(7676), pp. 354–359. doi: 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. (2019) 'Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model'. doi: 10.1038/s41586-020-03051-4.

[23] Yu, C., Dong, Y., Li, Y. and Chen, Y. (2020) 'Distributed multi-agent deep reinforcement learning for cooperative multi-robot pursuit', *The Journal of Engineering*. Institution of Engineering and Technology (IET), 2020(13), pp. 499–504. doi: 10.1049/joe.2019.1200.

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

[25] Tan, M. (1997) 'Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents', in *Readings in Agents*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp. 487–494. doi: 10.5555/284680.284934.

[26] Corne, D., Reynolds, A. and Bonabeau, E. (2012) 'Swarm Intelligence', in *Handbook of Natural Computing*, pp. 1599–1623.

[27] Espinós Longa, M., Tsourdos, A. and Inalhan, G. (2022) 'Human-Machine Network through Bio-inspired Decentralized Swarm Intelligence and Heterogeneous Teaming in SAR Operations', *Journal of Intelligent and Robotic Systems (submitted)*.

[28] Espinós Longa, M., Inalhan, G. and Tsourdos, A. (2022) 'Swarm Intelligence in Cooperative Environments: Introducing the N-Step Dynamic Tree Search Algorithm', in *AIAA SCITECH 2022 Forum*. Reston, Virginia: American Institute of Aeronautics and Astronautics, pp. 1–13. doi: 10.2514/6.2022-1839.

[29] Abed-alguni, B. H., Chalup, S. K., Henskens, F. A. and Paul, D. J. (2015) 'A multi-agent cooperative reinforcement learning model using a hierarchy of consultants, tutors and workers', *Vietnam Journal of Computer Science*. Springer Science and Business Media LLC, 2(4), pp. 213–226. doi: 10.1007/s40595-015-0045-x.

[30] Duman, E., Kaya, M. and Akin, E. (2005) 'A multi-agent fuzzy-reinforcement learning method for continuous domains', in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pp. 306–315. doi: 10.1007/11559221_31.

[31] Huang, J., Yang, B. and Liu, D.-Y. (2005) 'A Distributed Q-Learning Algorithm for Multi-Agent Team Coordination', in *Machine Learning and Cybernetics*, pp. 108–113. doi: 10.1109/ICMLC.2005.1526928.

[32] Ishiwaka, Y., Sato, T. and Kakazu, Y. (2003) 'An approach to the pursuit problem on a heterogeneous multiagent system using reinforcement learning', *Robotics and Autonomous Systems*, 43(4), pp. 245–256. doi: 10.1016/S0921-8890(03)00040-X.

[33] Kuremoto, T., Tsurusaki, T., Kobayashi, K., Mabu, S. and Obayashi, M. (2013) 'An improved reinforcement learning system using affective factors', *Robotics*. MDPI AG, 2(3), pp. 149–164. doi: 10.3390/robotics2030149.

[34] Wang, Y., Dong, L. and Sun, C. (2020) 'Cooperative control for multi-player pursuit-evasion games with reinforcement learning', *Neurocomputing*. Elsevier B.V., 412, pp. 101–114. doi: 10.1016/j.neucom.2020.06.031.

[35] Daoui, C., Abbad, M. and Tkiouat, M. (2010) 'Exact Decomposition Approaches for Markov Decision Processes: A Survey', *Advances in Operations Research*. Edited by I. Kacem. Hindawi Publishing Corporation, 2010, p. 19. doi: 10.1155/2010/659432.

[36] Kingma, D. P. and Ba, J. (2014) 'Adam: A Method for Stochastic Optimization', *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–15. Available at: http://arxiv.org/abs/1412.6980.