Proceedings of AIAA SCITECH 2022 Forum, 3-7 January, 2022, San Diego, CA and Virtual Event
Paper number AIAA 2022-1839, DOI:10.2514/6.2022-1839

Swarm Intelligence in Cooperative Environments: Introducing the N-Step Dynamic Tree Search Algorithm

Marc Espinós Longa*, Antonios Tsourdos† and Gokhan Inalhan‡
Cranfield University School of Aerospace, Transport & Manufacturing (SATM), Cranfield, Bedfordshire, MK43 0AL, United Kingdom

Uncertainty and partial or unknown information about environment dynamics have led reward-based methods to play a key role in the Single-Agent and Multi-Agent Learning problem. Tree-based planning approaches such as Monte Carlo Tree Search algorithm have been a striking success in single-agent domains where a perfect simulator model is available, e.g., Go and chess strategic board games. This paper presents a decentralized tree-based planning scheme, that combines forward planning with direct reinforcement learning temporal-difference updates applied to the multi-agent setting. Forward planning requires an engine model which is learned from experience and represented via function approximation. Evaluation and validation are carried out in the Hunter-Prey Pursuit cooperative environment and performance is compared with state-of-the-art RL techniques. *N-Step Dynamic Tree Search* (NSDTS) pretends to adapt the most successful single-agent learning methods to the multi-agent boundaries in a decentralized system structure, dealing with scalability issues and exponential growth of computational resources suffered by centralized systems. NSDTS demonstrates to be a remarkable advance compared to the conventional Q-Learning temporal-difference method.

I. Nomenclature

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

* PhD Researcher, School of Aerospace, Transport & Manufacturing
† Head of Center, Director of Research, School of Aerospace, Transport & Manufacturing, and AIAA Senior Member
‡ BAE Systems Chair, Deputy Head of Center, School of Aerospace, Transport & Manufacturing, and AIAA Associate Fellow

Published by AIAA. This is the Author Accepted Manuscript issued with: Creative Commons Attribution Non-Commercial License (CC:BY:NC 4.0).
The final published version (version of record) is available online at DOI:10.2514/6.2022-1839. Please refer to any applicable publisher terms of use.

II. Introduction

Machine Learning has become one of the greatest hot topics in the industry over the last decades due to its huge range of application. There are typically three different branches to approach a learning problem: *supervised*, *unsupervised* and *reward-based* learning methods [1]. Supervised learning generally uses function approximation techniques to reach desired outputs via direct feedback, and has proven to be a natural fit for classification tasks [2] [3]. Oppositely, unsupervised methods aim to recognize patterns in data [4] [5] without any feedback whatsoever. Addressing the single-agent and multi-agent problems frequently involves partial or no information about the boundary conditions and high uncertainty levels. Such configuration does not allow to work with systematic data structures from supervised methods, and often requires certain degrees of feedback to succeed. Reward-based learning algorithms are a sound success in this research area due to the inherent ability to generate unique solutions with reinforcements and data restrictions.

There are two major subsets under the reward-based umbrella: *stochastic search* algorithms which directly learn behaviors and policies without appealing value functions (i.e., estimates of successor states or state-action pairs), and *Reinforcement Learning* (RL) methods that make use of these under a *Markov Decision Process* (MDP) formalization. Evolutionary computation is an active field within the first group based on Darwinian models of evolution applied to refine populations of candidate solutions. *Genetic Programming* (GP) [6] and *Coevolutionary Algorithms* (CEAs) [7] [8] [9] are remarkable approaches adopted in this branch. While evolutionary computation can be efficient in small state spaces, RL techniques develop more rigorous policies considering states and actions taken at every iteration step; thus, despite having a higher demand of computational resources, the inferred solutions tend to be more optimal.

The RL spectrum comprehends a great amount of design features that can be considered when building learning agents under a MDP frame. From sample *temporal-difference* (TD) methods [10] [11] [12] [13] that make one-step updates of state or state-action values based on direct interaction with the environment, to n-step algorithms that wait a variable or fixed number MDP transitions to make value updates [14] [15]. While TD techniques present faster learning rates, they suffer from bias due to bootstrapping (estimating values in function of future value estimates). Differently Monte Carlo methods, the maximum expression of n-step algorithms, wait until episode termination to carry out updates, removing any bias from the process at expense of augmenting variance. Algorithm possibilities are expanded in the wide dimension when defining updates. Expectations [16] contemplate all available trajectories for a given agent behavior (policy), achieving balanced value estimates at a higher computational price.

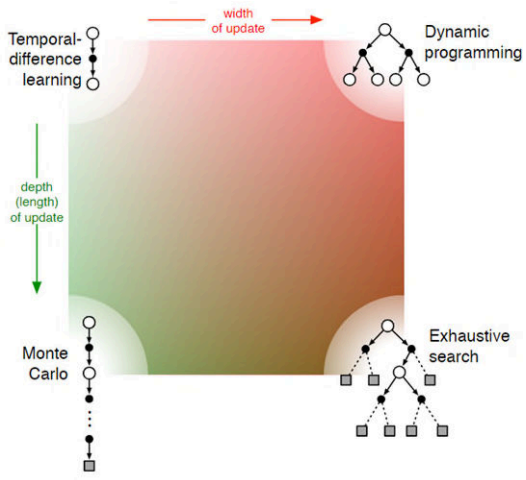

Fig. 1 Simplified map of sample-based learning methods [17]

Besides features in Fig. 1, additional aspects are worth considering when designing learning agents. Planning is a powerful form to accelerate learning by alternating direct RL updates from real environment interactions and simulated trajectories (model methods), either coming from agent experience using replay buffers [18] [19] or from built models [20]. Tree search methods have gained a lot of popularity in the single-agent domain after 2016, when AlphaGo Tree Search program defeated the 18-time world champion Go player [21]. This exhaustive search approach uses a model to do forward planning steps, making predictions to optimize action selection mechanisms. *Monte Carlo Tree Search* algorithm (MCTS) is a standard in this collective, where different sample trajectories are evaluated in depth on early

steps, and value estimates are computed by selecting greedily one trajectory and reaching a terminal stage following a predefined agent policy (rollout policy). Many fruitful variants of MCTS have recently emerged, including *MuZero* algorithm developed by DeepMind, validated and tested over an assortment of Atari games accomplishing superhuman performance [22].

Transitioning from single-agent to multi-agent framework is no trivial task. Even from a *team learning* perspective where agents are treated as a master individual and single-agent techniques can directly be applied, big issues emerge related to incredibly large 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 depends on a leading node which is a single point of failure to the system. *Concurrent learning* is a distributed network of agents that handles individual learning processes in parallel [23] [24]. Hence, agents need to co-adapt and find optimal cooperative behaviors from individual and global reinforcements. Assigning efficient rewards to individuals is part of the credit assignment problem and remains a challenge in the *Multi-Agent Reinforcement Learning* (MARL) problem. Likewise, intern communication between operative devices has a direct impact on performance. While some approaches adopt direct communication sharing TD-updates, policies or even full episodes [25], others opt for indirect bio-inspired mechanisms based on *stigmergic* behaviors [26] [27] or no communication at all.

This piece of research attempts to bring the most successful single-agent RL algorithms to the multi-agent domain in a decentralized fashion. Inspired by MCTS and n-Step Tree Backup [17] update, the N-Step Dynamic Tree Search (NSDTS) combines forward planning and direct RL at the same time, achieving a massive performance breakthrough compared to classic RL techniques such as Q-Learning. Forward tree search is executed via a learned neural network model trained with real environment experience to improve swarm action selection mechanisms. Communication is neglected to ensure system robustness, although future versions might consider data sharing to enhance performance. Thusly, distributed predictions are carried out assuming *best play* of agents. Expected updates are utilized to weight state-action value estimates, stored in individual *q tables*, and agents are reinforced with hybrid rewards (individual and global reinforcements) to successfully develop cooperative behaviors from parallel learning processes. The developed algorithm is tested and validated in the hunter-prey pursuit environment against distributed Q-Learning TD method. The pursuit game has been a reference to MARL problem since almost its existence [25] [28] [29] [30] [31] [32] [33] [23] [24], involving cooperation from a group of learning agents that, following a pursuer role, try to capture one or multiple evaders. As a result of the tabular setting, this paper contemplates a grid world with only two intelligent hunters and a random behavior prey, although future research includes transitioning to the function approximation sphere.

The rest of the paper is organized as follows: Section III provides a theoretical background in MARL, including MDPs, Q-Learning TD method and the new NSDTS algorithm. A general view of the developed system is presented in Section IV, alongside a description of the NSDTS models used to carry out forward predictions. Section V formulates the hunter-prey pursuit problem, incorporating specific environment features, rules of the game and the adopted reward function structure to train agents. Hyperparameter analysis and algorithm performance experimentation results and discussion are exposed in Section VI. Summary, concluding marks and future work suggestions are given in Section VII.

III. Theoretical Background

NSDTS and Q-Learning algorithms are built in an MDP framework and share common RL features. Therefore, this section includes a short introduction to RL where basic concepts such as MDP, return or state-action value estimates are described. Thereafter, Q-Learning and n-Step Dynamic Tree Search algorithms are presented.

A. Markov Decision Process

MDPs formalize the problem of one or multiple agents interacting with the environment. Nevertheless, single-agent and multi-agent frames present slight variations. In the MARL MDP setting, 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 $S_{t+1} = (S_{t+1}^0, S_{t+1}^1, \dots, S_{t+1}^{n-2}, S_{t+1}^{n-1})$ and gives R_{t+1} joint rewards to every learning agent. Each cycle represents a MDP step, and episodes are step sequences $< S_0, A_0, R_1, S_1, A_1, \dots, R_{T-1}, S_{T-1} >$ that at time $t = T$ reach a terminal joint state.

The aim of RL agents is to maximize rewards in the long-term. Hence, there must be a balance between immediate rewards and longstanding consequences of actions. The expected discounted sum of future rewards, also known as return, is a common approach to solve this dilemma.

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

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

Problem conditions define the basic characteristics of MDPs. *Partially-Observable Markov Decision Processes* (POMDPs) restrict agents from fully knowing MDP joint states because of sensor limitations, environmental factors or just experimental conditions [25]. Other assignments allow state aggregation and abstraction, allowing different parts of the problem to be processed independently [34]. This is the case of *factored* MDPs (FMDPs). As exposed in Section V Problem Formulation, the study carried out in this paper is built under a fully-observable MDP, i.e., agents can see all current states without restrictions.

B. Q-Learning

Q-Learning belongs to the family of TD methods within the realm of sample-based learning. State-action value estimates $Q^i(S_t, A_t^i)$ are used to numerically determine the relevance of agent i performing certain action A_t^i in S_t joint states at t time step. Every intelligent agent i makes, updates, and stores its own estimates in tables (tabular setting) and uses them subsequently to improve policies.

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

Equation (2) displays the incremental update rule generally used by sample-based learning methods, including TD and Monte Carlo algorithms, where α is the step size parameter (also known as learning rate). $\hat{G}_t^i$ return estimate is the target of the update, and the difference between $\hat{G}_t^i$ and the old state-action value is called TD error. Thus, α moderates the influence of TD error in the update.

The recursive form of Bellman equations defines a relationship between state-action values and their possible successors. Sample-based learning is inspired in these expressions to compute $\hat{G}_t$. For a given policy π^i (agent behavior) followed by agent i, the Bellman state-action function, i.e., the expected return given state s and action a, can be computed as follows.

$$q_\pi^i(s,a) \doteq \mathbb{E}_{\pi^i}\left[G_t^i \middle| S_t = s, A_t^i = a\right] = \sum_{s'}\sum_{r} p(s', r|s, a)\left[r^i + \gamma \sum_{a'} \pi^i(a'|s') q_\pi^i(s', a')\right] \tag{3}$$

Where $p(s', r|s, a)$ is the probability of transitioning to next joint states s' and receive r^i reward given s and a. The expectation term $\sum_{a'} \pi^i(a'|s') q_\pi^i(s', a')$ of Eq. (3) bootstraps multiplying the probability of selecting next action given s' next MDP state, by its respective state-action value for each a' next available action.

Sample-based learning algorithms like Q-Learning lack of an environment model, and thus are not able to compute $p(s', r|s, a)$ dynamics term. Instead, sampled trajectories from real interactions are used to update state-action values. Specifically, Q-Learning exploits the maximum next state-action value registered to make updates, as referred in Eq. (4).

$$\hat{G}_t^i = R_{t+1}^i + \gamma \max_{a'} Q^i(S_{t+1}, a') \tag{4}$$

Note that $\hat{G}_t^i$ does not depend on policy π^i. Off-policy methods, like Q-Learning, can adopt exploratory behaviors without affecting state-action value updates, which can be a powerful technique to balance exploration and exploitation. Another common approach to achieve that balance in RL is ϵ-greedy action selection policy.

$$A_t^i \leftarrow \begin{cases} \underset{a}{\operatorname{argmax}}\, Q_t^i(S_t, a) & \textbf{with probability } 1-\epsilon \\ a \sim uniform(\{a_0, a_1, \ldots, a_k\}) & \textbf{with probability } \epsilon \end{cases} \tag{5}$$

Remark that Eq (5) is designed for small $\epsilon > 0$. Greedy actions are selected with probability $1 - \epsilon$, while an exploratory action is taken randomly with probability ϵ. This study uses an ϵ-greedy policy π for every agent.

C. N-Step Dynamic Tree Search

Inspired by MCTS and n-Step Tree Backup algorithms, NSDTS powers two engines at the same time: a forward planning mechanism that helps agents to make predictions and improve action selection, and a real environment interaction that weights, corrects, and incorporates additional updates based on experience.

The trained model used to execute forward planning can be decomposed in two submodules: a probabilistic behavior model and an environment model. The first one is in charge to predict evader actions based on the current joint states S_t. After predicting the next prey move, the environment model determines S_{t+1} joint states transition selecting the greediest individual action A_t^i and assuming best play of swarm mates according to $Q^i(S_t, A_t^i)$. Once generated N future samples, updates are backpropagated until reaching the root node of the tree search (time step t).

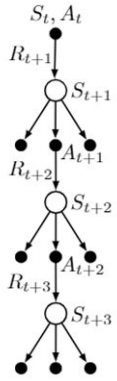

Fig. 2 N-Step Tree-Backup update for 3 steps [17]

As exposed in Fig. 2, updates not only do consider all possible action trajectories, but also future samples. Hence, the estimated return for N steps ahead $\hat{G}^i_{\tau:\tau+N}$ is reflected in Eq. (6).

$$\hat{G}^i_{\tau:\tau+N} = R^i_{\tau+1} + \gamma \sum_{a \neq A^i_{\tau+1}} \pi^i(a|S_{\tau+1}) Q^i_{\tau+N-1}(S_{\tau+1}, a) + \gamma \pi^i\left(A^i_{\tau+1} \middle| S_{\tau+1}\right) \hat{G}^i_{\tau+1:\tau+N} \tag{6}$$

Notice the difference between real time t and forward fictitious time τ. As state-action value updates are implemented progressively following Eq. (2) TD incremental update rule, Q^i table of agent i will first be refreshed at $\tau + N - 1$ time step, thence the subscript. It is worth noting that for the special cases where $\tau + 1 = N$ and $\tau + 1 = T$, being $\tau = \min(T, t + [0,1, \dots, N-1])$, return is computed either like an expected TD update

$$\hat{G}^i_\tau = R^i_{\tau+1} + \gamma \sum_a \pi^i(a|S_{\tau+1}) Q^i(S_{\tau+1}, a) \tag{7}$$

Or just $\hat{G}^i_\tau = R^i_T$ respectively. Equation (7) coincides with Expected SARSA TD algorithm return, which balances state-action values with π^i probability distribution. Remark that Eq. (6) is the multi-step version of Eq. (7).

After backpropagating all returns, agent i selects an action according to π^i and interacts with the real world. Table 1 discloses the full NSDTS algorithm step by step.

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, $s \in \mathcal{S}$ joint states and $a \in \mathcal{A}(s)$
Initialize $net(s)$ probabilistic behavior model and $Model(s,a)$ environment model
Algorithm parameters: step size $\alpha \in (0,1]$, small $\epsilon > 0$ for policy π^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_τ prey action with $net(S_\tau)$
      $A^i_\tau, R^i_{\tau+1}, S_{\tau+1} \leftarrow Model(S_\tau, a_\tau)$ 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 ($i = 0,1,\dots,n-1$):
    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:
      $G^i_t \leftarrow R^i_{t+1}$
    $Q^i(S_t, A^i_t) \leftarrow Q^i(S_t, A^i_t) + \alpha[G^i_t - Q^i(S_t, A^i_t)]$

Observe that the equations used at line 24 and 27 to make real world updates are Eq. (7) and Eq. (2). NSDTS improves action selection and increments the learning rate of agents by combining future sample trajectories and real-world updates.

IV. System Architecture

For further comprehension and reproducibility, it is worth having an overall view of the developed NSDTS system as well as a closer examination to the implemented models. Figure 3 diagram exposes the architecture of the deployed system, including the different models and interaction sources exploited by the algorithm.

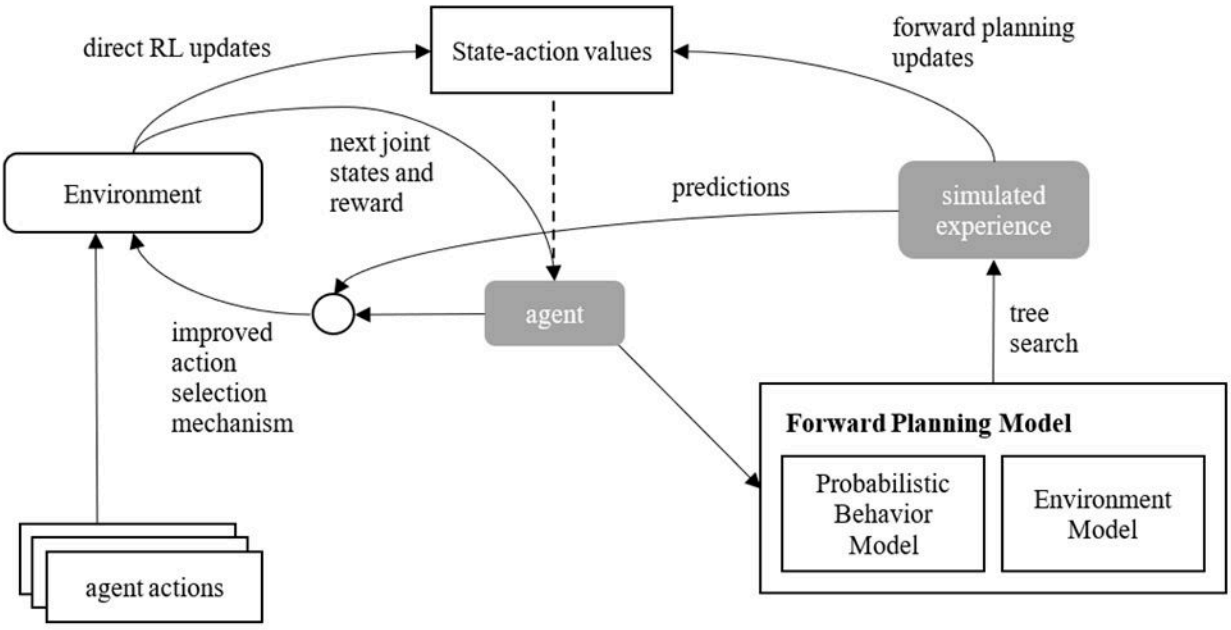

Fig. 3 NSDTS System overview from an individual agent perspective

Agents have direct access to their individual Q tables and use them either for forward planning or direct updates. Notice the full-observability of the MDP in Fig. 3, i.e., each agent can sense all joint states transitions from the environment. Nevertheless, actions taken by teammates are assumed best and thus, can differ from real actions. This assumption is taken to fulfil a design premise: system robustness. The swarm must be able to work even under no communication circumstances. Swarming performance enhancements through data exchange and team agreements are considered future work.

The probabilistic behavior model consists of a neural network that predicts the evader's next action given S_t joint states, based on a learned probability distribution. Figure 4 presents the conceived configuration of the prediction model.

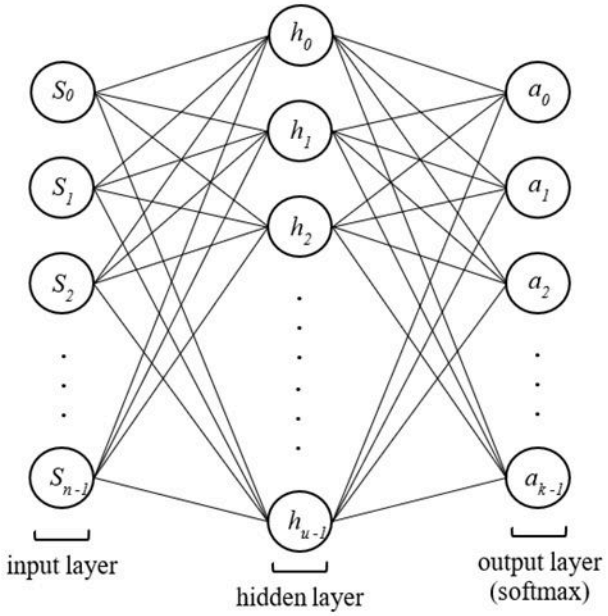

Fig. 4 Probabilistic behavior model

Notice that the network has as many input neurons as joint MDP states S_t. The number of hidden neurons u remains a hyperparameter, while the output layer is formed by all the available actions k that an agent can execute. As aforementioned, this model predicts the next prey agent action based on a resulting probability distribution, thence the softmax output layer. This can also be contemplated as a classifier problem, where the input is a collection of MDP states, and the output is an action class; thereupon, a cross entropy function is used to compute the error between the output and the ground truth data. However, remark that the interest focuses on the probability distribution rather than the class itself to foresee all possible scenarios.

Optimization of the neural network is executed via backpropagation[§] using ADAM method [35]. This advanced form of *Stochastic Gradient Descent* (SGD) uses adaptive vector step sizes, ergo, greater learning rates dedicated to parameters with higher errors, and low-order moments that progressively boost the weight gradients towards continuous directions. Altogether, these techniques improve the learning process and reach global or local minimums faster. The network is trained with a stochastic batch size, i.e., weights are modified on every sample of the training dataset.

V. Problem Formulation

The hunter-prey pursuit problem has been used as a benchmark in MARL for years. Continuous state spaces [33] [31], grid domains [24] [25] and obstacles [23] are some of the modified environment features that authors have considered in their respective research. This section contains a description of environment key aspects and fundamental rules that define the experimentation setup.

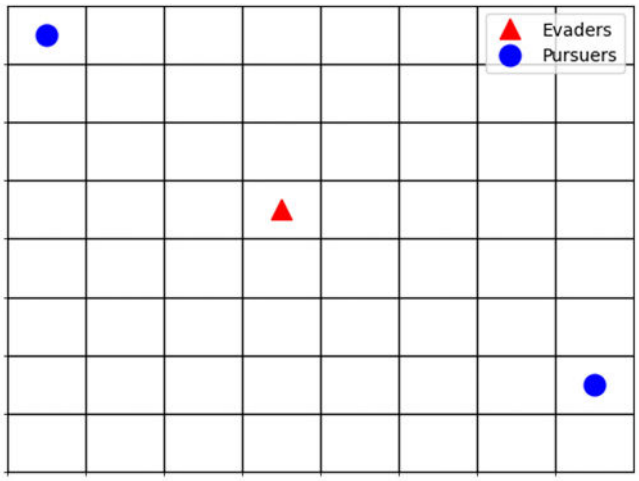

Fig. 5 Hunter-prey pursuit game environment

Figure 5 reveals an 8 by 8 grid world with an evader and two intelligent pursuers, which equals to a total of 1,310,720 MDP states for the given representation. At the beginning of each episode, the prey is initialized at the center of the grid, while hunters are randomly allocated in non-terminal states. Pursuers must capture the evader agent without colliding. Individual captures occur when a hunter and a prey occupy the same cell, while cooperative captures arise if two or more pursuers are in the nearest consecutive cells of the evader, just like Fig. 6 refers.

[§] Optimization technique that computes the gradient of the loss function (error) with respect to the function approximation parameters.

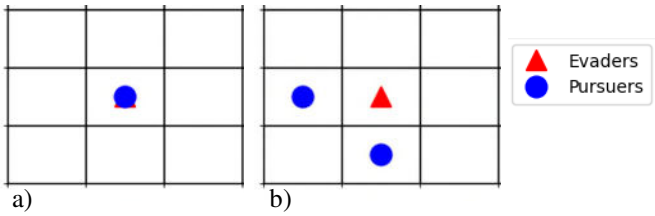

Fig. 6 Capture examples: a) individual capture, b) cooperative capture

To encourage cooperation, captures in group receive higher rewards than individual captures. On the other hand, collisions happen either if two or more hunters occupy the same cell, or if a hunter placed on the edge of the grid takes an action towards the wall. Anyhow, agents involved in the crash are penalized with a negative reward. Furthermore, agents are given a negative compensation for every time step of the episode to stimulate hunting efficiency. The available actions for each agent are going up, right, down, and left directions, moving to the respective contiguous cells. An agent can also decide to remain on the same cell given the occasion. Evaders' action selection mechanism is a uniformly random policy, although future work contemplates intelligent behaviors.

VI. Experimental Results and Discussion

This section compares the performance of NSDTS algorithm and Q-Learning in the MARL domain. The experimentation is carried out in the hunter-prey pursuit world described in Section V Problem Formulation. However, it is worth noting that a previous hyperparameter analysis is required as far as the learning rate parameter α is concerned. A bad step size selection can heavily impact the performance of both approaches. Table 2 summarizes the setup configuration adopted to carry out the respective study.

Table 2 Hyperparameter analysis configuration

Algorithm Parameters	
Planning steps (N)	5
Discount factor (γ)	1.00
Exploring parameter (ϵ-greedy policy)	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)	$9 \cdot 10^{-2}$
Second moment update parameter (β_v)	$9.99 \cdot 10^{-2}$
Batch size	1
Training samples	5,000,000
Testing samples	1,000,000
Model cumulative error	$4.2945 \cdot 10^{-4}$
Hybrid Reward Function	
Colliding with boundaries or other agents	-100
Each step	-1
Individual captures	$+10$
Cooperative captures	$+100 \times n_c$

Notice that Table 2 algorithm parameters are applied to both Q-Learning and NSDTS methods, except from N that only can be applied to the latter. Neural network learning rate (not to confuse with α learning rate parameter from RL algorithms) determines the weight gradient size towards a minimum of the loss function, while λ weight decay is used

to avoid overfitting. The mean moment update β_m and second moment update β_v regulate the adaptive vector step sizes and low order moments of ADAM optimizer. As far as the hybrid reward function is concerned, observe that a combination of individual and global rewards has been designed to solve the MARL credit assignment problem, being n_c the number of intelligent agents involved in the cooperative capture.

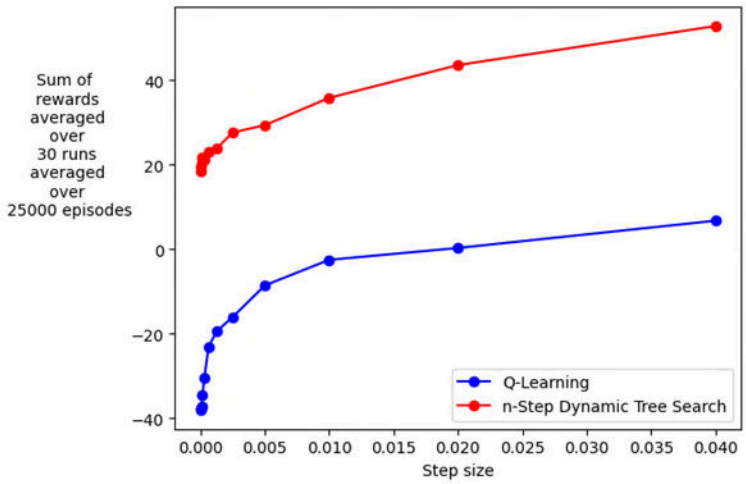

Fig. 7 Hyperparameter analysis results

Figure 7 presents the average performance of Q-Learning and NSDTS algorithm for 5 planning steps and different step sizes. Optimality is reached for higher α values in both cases. Despite being a hyperparameter analysis, it is already noticeable the difference between NSDTS and Q-Learning. After 25000 episodes averaged over 30 runs, for a learning rate of $\alpha = 0.040$, NSDTS's overall performance is almost 8 times better than Q-Learning, which represents an improvement of 800%.

Further experimentation is carried out, following the same configuration displayed in Table 2, to further comprehend the behavior of agents throughout each episode. The step size value used in this case corresponds to the optimal value shown by Fig. 7, i.e., $\alpha = 0.040$. Figure 8 exposes the sum of rewards for 25,000 episodes averaged over 30 runs. Observe that NSDTS algorithm has been studied for 2 and 11 forward planning steps.

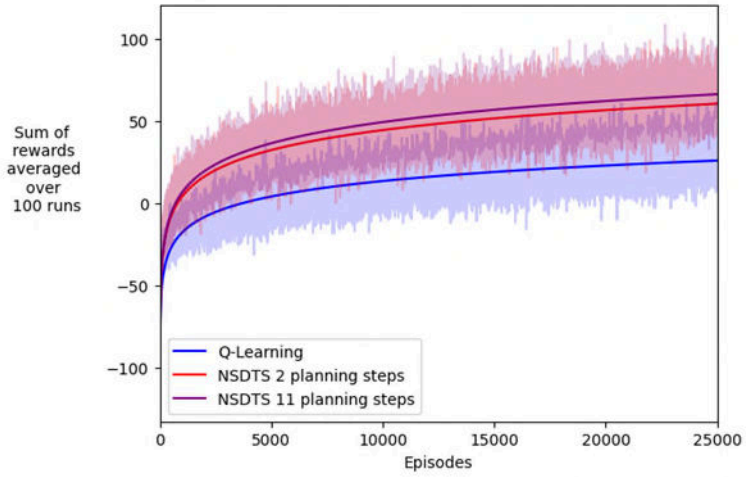

Fig. 8 Q-Learning and NSDTS averaged rewards for different planning steps

NSDTS algorithm demonstrates not only higher averaged rewards, but also a greater learning rate. During the first 1000 episodes, n-Step Dynamic Tree Search learns much faster than Q-Learning before converging to a near optimal behavior. For the given episodes, NSDTS best policy for only 2 planning steps outperforms Q-Learning by far. Although 11-Step Dynamic Tree Search achieves the best performance, the improvement is rather small compared to

the 2-step version. As previously mentioned in Section V Problem Formulation, evader's action selection mechanism is completely random. Therefore, despite planning forward many steps ahead, there is an intrinsic error linked to prey policy's unpredictable nature, i.e., the 2-step forward planning method converges to a nearly optimal policy, such as the 11-step search. Increasing the complexity of the problem (e.g., developing intelligent behaviors for prey agents) may require further planning steps to obtain an optimal policy, and thus the difference between 2-step and 11-step forward planning methods would be greater.

Both Fig. 7 and Fig. 8 rewards have been averaged as well over pursuer agents. Referring to the hybrid reward function described in Table 2, it is easily deducible that NSDTS agents develop often cooperative behaviors, as the averaged reward is superior to 10, which corresponds to the obtained value for individual captures. At episode 25,000, NSDTS hunters attain 60 and 70 averaged rewards for 2 and 11 forward planning steps respectively. On the contrary, although Q-Learning agents evolve cooperation as well (the averaged reward at episode 25,000 is approximately 25), agents struggle to avoid collisions, thence their lower rewards. This effect can further be contemplated in Fig. 9, where the number of steps required to terminate an episode averaged over 100 runs is shown for 50,000 episodes.

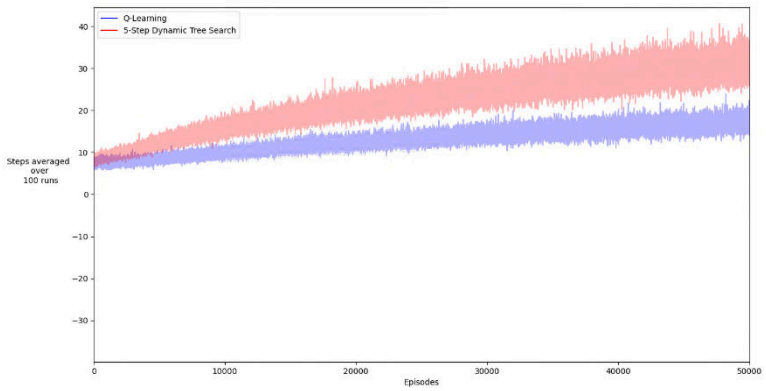

Fig. 9 Q-Learning and 5-Step Dynamic Tree Search averaged steps per episode

While Q-Learning reaches termination with 18 steps towards the 50,000 episode, 5-Step Dynamic Tree Search takes 32 steps to hit a terminal state. Even though every step is penalized with -1, NSDTS averaged rewards are higher than Q-Learning rewards, just as Fig. 7 and Fig. 8 state. In other words, NSDTS agents intelligently execute their movements to avoid collisions, achieving frequent cooperative captures regardless of the unexpected random evader's policy, while Q-Learning hunters fail in this aspect.

VII. Conclusion and Future Work

This paper attempts to bring the most outstanding single-agent RL techniques to the multi-agent setup. Inspired by single-agent tree search methods that recently achieved superhuman performance in many Atari games, and classic board games such as chess and Go, the developed N-Step Dynamic Tree Search algorithm represents a significant advance in the multi-agent domain compared to conventional MARL techniques such as Q-Learning. Inferred model predictions enables NSDTS not only to achieve a striking performance, but also to acquire knowledge at much faster rates. While NSDTS presents an improvement of 300% with respect to Q-Learning agents after policy convergence, the overall performance surpasses Q-Learning by 800%.

The experimentation in this work, carried out within the frame of the hunter-prey pursuit problem, is constructed on a tabular setting where each agent stores state-action values in decentralized tables. Despite developing cooperative behaviors, agents do not exchange any type of information; instead, they assume *best play* of every agent involved in the game according to their decentralized source of knowledge. The testing scenario is an 8 by 8 grid-world with an evader that follows a uniformly random policy, and two intelligent pursuers. Future work consists in transitioning from tabular to function approximation domain to scale up the experiment, adding complexity to the environment with more agents (evaders and pursuers), bigger continuous worlds with obstacles, etc. Intelligent prey behavior and information exchange between agents is also considered.

Acknowledgments

This work is sponsored by the Engineering and Physical Sciences Research Council (EPSRC) and BAE Systems under the project reference no. 2454254.

References

[1] 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.

[2] 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: Springer Berlin Heidelberg, pp. 21–49. doi: 10.1007/978-3-540-75171-7_2.

[3] 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.

[4] 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.

[5] 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.

[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] 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.

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

[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. (2021) 'Human-Machine Network through Bio-inspired Decentralized Swarm Intelligence and Heterogeneous Teaming in SAR Operations'.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

https://dspace.org/

Swarm intelligence in cooperative environments: introducing the N-step dynamic tree search algorithm

Espinós Longa, Marc; Inalhan, Gokhan; Tsourdos, Antonios

2021-12-29
AIAA
https://doi.org/10.2514/6.2022-1839
Espinós Longa M, Inalhan G, Tsourdos A. (2021) Swarm intelligence in cooperative environments: introducing the N-step dynamic tree search algorithm. In: AIAA SciTech 2022 Forum, 3-7 January 2022, San Diego, CA, USA and Virtual Event, Paper number AIAA 2022-1839

Downloaded from a DSpace repository. https://dspace.org/