

Offline Nash Solvers Meet Online Tree Search in Multi-Agent Games on Graphs

Mukesh Kumar*, Yue Guan, and Panagiotis Tsiotras

School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA, USA
{mkumar361,yguan44,tsiotras}@gatech.edu

Abstract. Computing Nash equilibrium policies in multi-agent Pursuit-Evasion games (PEG) is challenging due to the exponential growth of the joint state and action spaces with the number of agents. Existing approaches either rely on offline equilibrium approximations, which may lack adaptability during execution, or online planning methods, which suffer from large branching factors. In this work, we propose Primitive-Guided Tree Search (PGTS), a hybrid framework that integrates offline exact Nash equilibrium computation with online tree search: PGTS first solves a collection of smaller, tractable sub-games offline; at deployment, PGTS performs online tree search at each time step, using the optimal sub-game policies and value functions to guide tree expansion and estimate leaf-node values. Extensive experiments on varied graph topologies, including real-world networks, demonstrate that PGTS significantly outperforms state-of-the-art learning and heuristic baselines, while maintaining robust performance against adversaries.

Keywords: Multi-Agent Systems · Pursuit–Evasion Games · Monte Carlo Tree Search.

1 Introduction

Multi-agent decision-making under adversarial conditions arises in a wide range of applications, including monitoring and surveillance, urban security, border patrol, and autonomous pursuit tasks [3,11]. A canonical model for studying such interactions in discrete settings is the graph-based *Pursuit–Evasion* game (PEG) [8,15]. The standard solution concept for such games is the Nash equilibrium. While the existence of equilibrium policies is well established [5], computing equilibria exactly in realistic multi-agent settings remains a significant challenge. As the number of agents increases, both the state space and the joint action space grow rapidly. This combinatorial growth renders exact equilibrium computation impractical even for moderately sized environments, motivating the development of scalable approximation and planning techniques.

Existing approaches to approximating solutions in multi-agent settings can be broadly classified into three categories: offline, online, and hybrid. Offline approaches, such as Policy Space Response Oracles (PSRO) [2] and its variants,

* Corresponding author.

provide a general framework for computing approximate equilibria by iteratively training best-response policies and maintaining a growing meta-game over the policy space. In the Pursuit-Evasion context, MT-PSRO [13] combines PSRO with multi-task pretraining, and uses MAPPO [23] as the best-response oracle. While offline methods enable fast decision-making during execution, their training complexity can grow quickly with the number of agents. Moreover, the agents’ performance can degrade significantly when they encounter adversarial behaviors that differ from those encountered during training.

Online planning methods, such as Monte Carlo Tree Search (MCTS) [19], adapt to changing situations and reason about strategic interactions at execution time by simulating a large number of future trajectories. Although originally introduced to solve sequential in-turn games, extensions of MCTS to simultaneous-move games have been explored through Simultaneous-Move MCTS (SM-MCTS) [20]. Unlike offline methods, online search methods can reason directly about the current game state, but their decision quality depends critically on the ability of the search process to identify and explore promising future trajectories. In multi-agent games, this exploration becomes increasingly difficult because the number of joint actions, and hence the branching factor, grows exponentially with the number of agents. As a result, practical implementations often rely on carefully designed heuristics, or domain-specific knowledge to focus the search on promising regions of the action space [6,14,21].

More recently, hybrid approaches have sought to combine the strengths of offline computation and online planning. Methods such as AlphaZero and its variants [1,16,18] have demonstrated that offline-computed priors can significantly improve the efficiency of online search. These methods focus primarily on two-player games, although similar ideas have also been explored in multi-agent settings. For example, NSGZero [22] combines neural policy and value function approximation with MCTS to learn Pursuer policies in network security games on graphs. While these approaches achieve strong empirical performance, they rely on learning accurate policy and value approximations, often requiring a substantial amount of training data.

Motivated by the strengths and limitations of existing approaches, we propose Primitive-Guided Tree Search (PGTS), a hybrid framework that combines offline game-theoretic computation with online planning. Rather than computing policy and value functions for the original multi-agent game directly, the offline phase of PGTS decomposes the original game into a collection of smaller, computationally tractable sub-games called *primitive sub-team games*, each involving only a subset of agents, and solves exactly these primitive sub-team games via value iteration. At deployment, PGTS performs online tree search and leverages the offline-computed solutions. Specifically, PGTS introduces two adaptations that make tree search more efficient and effective than existing variants. First, the equilibrium policies of the primitive sub-team games are used to guide *joint-action* rollouts, allowing the search to focus its computational effort on the most relevant regions of the joint action space. Second, the value functions of the primitive sub-games are used to approximate the leaf-node values

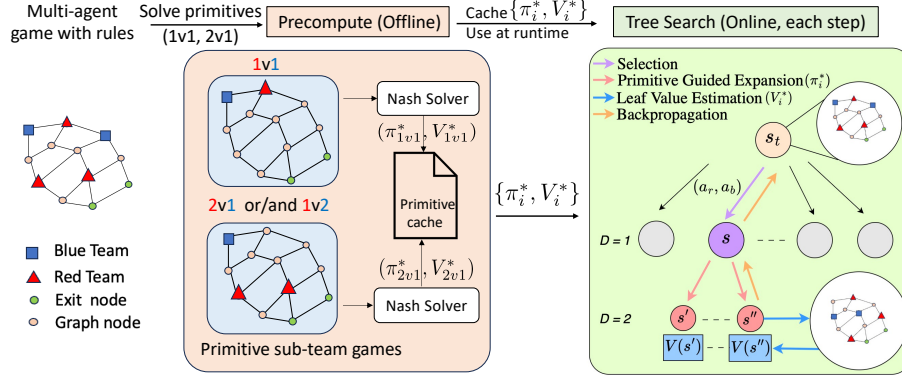


Fig. 1: Illustration of the hybrid approach for Pursuit-Evasion games. **Left:** a game configuration with three Pursuers (Red), two Evaders (Blue), and two exit nodes (Green). **Right:** pipeline of the proposed PGTS algorithm. Primitive sub-team games (e.g., 1v1, 2v1) are solved offline to obtain equilibrium policies and value functions. Online tree search iterates among four steps: **Selection**, **Expansion**, **Leaf Value Estimation**, and **Backpropagation**. The primitives computed offline are utilized to guide roll-out during **Expansion** and to perform **Leaf Value Estimation**.

of the MCTS search tree, allowing the search to terminate at a shallower depth while still maintaining good estimates of future game outcomes. Together, these adaptations keep both the branching factor and the effective search depth manageable, while preserving strong empirical performance. At the same time, since PGTS rolls out actions in the full joint action space, it retains the ability to reason about coordinated policies among all agents, thereby compensating for the lack of team-level coordination of the primitive games, which involve only a subset of the agents. Figure 1 illustrates the core idea of the hybrid algorithm for a multiple-agent Pursuit-Evasion game.

The remainder of the paper is organized as follows. Section 2 presents the formulation of the multi-agent Pursuit-Evasion game (PEG) on a graph, introduces the notion of Nash equilibrium, and presents the corresponding value-iteration algorithm for its computation. Section 3 presents the decomposition of the full game into primitive games involving subsets of agents. Section 4 develops the PGTS algorithm while Section 5 provides a comprehensive empirical analysis of PGTS under different PEG configurations, compares it with baseline algorithms, and presents ablation studies to demonstrate the contribution of each component. Finally, Section 6 concludes the paper with a summary of possible future extensions.

2 Problem Formulation

We consider a graph-based multi-agent Pursuit-Evasion game between a Red team of Pursuers and a Blue team of Evaders. The game is defined by the tuple $\mathcal{G} = \langle \mathcal{N}, G, V_e, \mathcal{P}, \mathcal{R}, \Gamma^G, \beta \rangle$, and the set of agents $\mathcal{N} = \mathcal{N}_r \cup \mathcal{N}_b$ is partitioned into the Red/Pursuers team, $\mathcal{N}_r$, consisting of $n_r = |\mathcal{N}_r|$ agents, and

the Blue/Evaders team, $\mathcal{N}_b$, consisting of $n_b = |\mathcal{N}_b|$ agents. The underlying environment is a graph $G = (V, E)$, where V denotes the set of vertices and $E \subseteq V \times V$ denotes the set of edges. The set $V_e \subseteq V$ denotes the set of exit vertices available to the Blue team. At a time step t , the positions of the Red and Blue agents are given by $s_t^r = (s_t^{r,1}, \dots, s_t^{r,n_r})$ and $s_t^b = (s_t^{b,1}, \dots, s_t^{b,n_b})$, respectively. The joint state is then $s_t = (s_t^r, s_t^b) \in V^{n_r} \times V^{n_b}$, and we denote this joint state space by $\mathcal{S}$. For each agent $i \in \mathcal{N}$, the available actions at vertex $s^i \in V$ are given by the set $\mathcal{A}^i(s^i) = \{v \in V \mid (s^i, v) \in E\} \cup \{s^i\}$, corresponding to either moving to an adjacent neighbor or staying in place. Consequently, for each team $* \in \{r, b\}$, the feasible action space at joint state $s = (s^r, s^b)$ is given by $\mathcal{A}^*(s^*) = \prod_{i \in \mathcal{N}_*} \mathcal{A}^i(s^i)$. At each time step, all agents act simultaneously. Given an action for agent i of the Red team at time t , we can define the (Red) team action as $a_t^r = (a_t^{r,1}, \dots, a_t^{r,n_r})$, and similarly for the Blue team, that is, $a_t^b = (a_t^{b,1}, \dots, a_t^{b,n_b})$. Given now a joint state s_t and a joint action $a_t = (a_t^r, a_t^b)$, the game transitions deterministically to the next state according to the transition function $s_{t+1} = \mathcal{P}(s_t, a_t)$. While, for the sake of simplicity, we focus on deterministic transitions in this work, stochastic transition kernels can be incorporated with only minor modifications within our approach. Furthermore, we assume a fully observable environment in which both teams have access to the complete joint state and full knowledge of the game dynamics.

The inter-team interactions are specified by the game rules Γ^G . Specifically, Γ^G defines the capture, escape, post-capture, and terminal conditions. In our setting, a Blue agent is captured whenever it lies within a capture radius of r_c hops from at least one Red agent. Similarly, a Blue agent escapes whenever it is within an escape radius of r_e hops from an exit node. The reward function $\mathcal{R}$ is defined at the team level, and is assumed to be zero-sum, with the Red team maximizing and the Blue team minimizing the expected cumulative reward, with $\beta \in (0, 1)$ serving as the discount factor; see equation (1) below. At any time step, the Red team receives a reward of +1 whenever a Blue agent is captured and a reward of -1 whenever a Blue agent escapes. The Blue team receives the corresponding opposite reward. The game terminates when all Blue agents have either been captured or escaped, or when another terminal condition specified by Γ^G is satisfied.

The game rule set Γ^G is sufficiently general to accommodate both finite and infinite-horizon formulations. To highlight the computational challenges inherent in solving the resulting multi-agent game $\mathcal{G}$, we focus the remainder of the discussion on the infinite-horizon setting, which provides a convenient framework for illustrating the scalability issues that motivate our approach.

2.1 Nash Equilibrium

We consider a Markovian policy π^* for each team $* \in \{r, b\}$, which is a mapping $\pi^* : \mathcal{S} \rightarrow \Delta(\mathcal{A}^*(s^*))$ from the joint state space to a distribution over valid moves at that team state. For notational convenience, we will also often write $\pi^*(a^*|s)$ to denote the probability assigned to team action $a^* \in \mathcal{A}^*(s^*)$ at joint state s . At any joint state s , the Red team selects an action $a^r \in \mathcal{A}^r(s^r)$, and the Blue

team selects an action $a^b \in \mathcal{A}^b(s^b)$. Given a pair of policies (π^r, π^b) for the two teams, the induced value for each state $s \in \mathcal{S}$ is defined as

$$\mathcal{V}^{\pi^r, \pi^b}(s) = \mathbb{E} \left[\sum_{t=0}^{\infty} \beta^t \mathcal{R}(s_t, a^r, a^b) \mid a^r \sim \pi^r, a^b \sim \pi^b, s_0 = s \right] \quad (1)$$

where $\beta \in (0, 1)$ is the discount factor and s_t denotes the joint state at time t . A Nash equilibrium is then defined as a pair of Markovian policies (π^{r*}, π^{b*}) such that, for all $s \in \mathcal{S}$ and all Markovian policies π^r, π^b , the following inequality holds:

$$\mathcal{V}^{\pi^r, \pi^{b*}}(s) \leq \mathcal{V}^{\pi^{r*}, \pi^{b*}}(s) \leq \mathcal{V}^{\pi^{r*}, \pi^b}(s), \quad s \in \mathcal{S}. \quad (2)$$

The overall objective is to compute the Nash equilibrium policies (π^{r*}, π^{b*}) that satisfy the saddle-point condition described in (2).

2.2 Shapley Value Iteration

The standard Shapley value iteration for two-player zero-sum stochastic games [4] can be utilized to solve the multi-agent game $\mathcal{G}$ by treating each team as a single decision-maker. To compute the equilibrium, Shapley value iteration updates are performed to the state-action (Q-value) matrices until convergence. Specifically, let $k \in \{0, 1, 2, \dots\}$ denote the iteration index, and let $\mathbf{Q}_k(s) \in \mathbb{R}^{|\mathcal{A}^r(s^r)| \times |\mathcal{A}^b(s^b)|}$ denote the Q-matrix for state s at iteration k . The update rule for this iterative process is given by

$$Q_{k+1}(s, a^r, a^b) = \mathcal{R}(s, a^r, a^b) + \beta \text{Nash}(\mathbf{Q}_k(s')), \quad (3)$$

where $\mathcal{R}(s, a^r, a^b)$ is the immediate reward to the Red team and $s' = \mathcal{P}(s, (a^r, a^b))$ is next state after applying the action pair (a^r, a^b) . The value $\text{Nash}(\mathbf{Q}_k(s'))$ denotes the Nash equilibrium payoff of the stage game defined by the matrix $\mathbf{Q}_k(s')$. From the maximizing Red team's perspective, the Nash value at any joint state s corresponds to the max-min value

$$\text{Nash}(\mathbf{Q}(s)) = \max_{\pi^r(s) \in \Delta(\mathcal{A}^r(s^r))} \min_{a^b} \sum_{a^r} \pi^r(a^r|s) Q(s, a^r, a^b), \quad (4)$$

which can be obtained by solving the following linear program:

$$\begin{aligned} & \max_{v, \pi^r(s)} v \\ & \text{s.t.} \quad v \mathbf{1} - \pi^r(s)^\top \mathbf{Q}(s) \leq 0, \\ & \quad \mathbf{1}^\top \pi^r(s) = 1, \quad \pi^r(s) \geq 0, \end{aligned} \quad (5)$$

where $\pi^r(s) \in \Delta(\mathcal{A}^r(s^r))$ is the Red team's policy and $\mathbf{1}$ is the vector of ones of compatible dimension. The solution v^* of (5) gives the Nash value $v^* = \text{Nash}(\mathbf{Q}(s))$, and $\pi^{r*}(s)$ gives the Nash policy for the Red team at joint state s . The Blue team policy can be obtained from the corresponding dual program or

directly from the dual variables of (5); see [4]. Under standard assumptions [4], the Shapley operator (3) is a contraction and the sequence $\{Q_k\}$ converges to a unique fixed point Q^* . For every state $s \in \mathcal{S}$, the corresponding optimal Nash value is then given by $V^*(s) = \text{Nash}(Q^*(s))$, and the equilibrium policy pair $\pi^* = (\pi^{r*}, \pi^{b*})$ is obtained by solving for the Nash equilibrium of the stage game induced by $Q^*(s)$.

While Shapley’s value iteration provides a principled solution method for two-player zero-sum stochastic games, each value update requires computing the Nash value of a local matrix game at each state. This makes a naïve adoption of Shapley’s value iteration to two-*team* zero-sum stochastic games computationally prohibitive. In particular, both the number of linear programs solved at each iteration and the size of the corresponding Q-function grow exponentially with the number of agents. As a result, exact computation quickly becomes intractable even for moderately sized teams and environments.

To address this challenge, we exploit the observation that many large-scale Pursuit–Evasion games are composed of recurring local interactions involving only a small number of agents. Rather than solving the full game directly, we first solve a collection of smaller representative games, referred to as *primitive sub-team games*, and use their equilibrium solutions as reusable building blocks. These *primitive sub-team games* involve only a small number of agents from each team (e.g., 1 Red vs. 1 Blue, 2 Red vs. 1 Blue, etc) and are played on the same graph with identical engagement rules Γ^G . Since these primitive games contain only a few agents, their exact Nash policies and value functions can be computed efficiently offline and reused during online execution.

A natural use of these primitive solutions is through a *sub-team decomposition* [7,17] where the original game is decomposed into a collection of primitive interactions and the resulting equilibrium policies are used directly for decision making. While computationally efficient, such an approach treats the primitive games independently and therefore captures only limited coordination across different groups of agents. To overcome this limitation, we propose *Primitive-Guided Tree Search* (PGTS), which incorporates the pre-computed primitive solutions into an online tree-search framework. The primitive policies guide exploration, while the primitive value functions provide estimates of future game outcomes. As a result, PGTS enables coordinated planning over the full agent team while preserving the computational advantages of the offline-computed primitive games.

We now describe the construction of these primitive sub-team games and the computation of their corresponding Nash equilibrium policies and value functions. To operationalize the offline phase, we compute exact Nash equilibrium solutions for two primitive sub-team games: 1 Red vs. 1 Blue (1v1) and 2 Red vs. 1 Blue (2v1); the approach is similar for subgames with more agents. Unlike the original multi-agent game, the restricted joint state spaces of the two subgames, namely, $\mathcal{S}_{1v1} = V \times V$ and $\mathcal{S}_{2v1} = V^2 \times V$, permit exact policy computation over the entire state space. Using the solver from [9], we derive the optimal value functions $V_{1v1}^* : \mathcal{S}_{1v1} \rightarrow \mathbb{R}$ and $V_{2v1}^* : \mathcal{S}_{2v1} \rightarrow \mathbb{R}$, along with

the corresponding Nash policy pairs $\pi_{1v1}^* = (\pi_{1v1}^{r*}, \pi_{1v1}^{b*})$ and $\pi_{2v1}^* = (\pi_{2v1}^{r*}, \pi_{2v1}^{b*})$. These policies map primitive states to probability simplices over joint actions. For instance, at state $s_{1v1} = (s^r, s^b) \in \mathcal{S}_{1v1}$, the policies $\pi_{1v1}^{r*}(s_{1v1}) \in \Delta(\mathcal{A}^r(s^r))$ and $\pi_{1v1}^{b*}(s_{1v1}) \in \Delta(\mathcal{A}^b(s^b))$ return the optimal mixed policy for the Red and Blue agents, respectively, with 2v1 policies scaling analogously over $\mathcal{S}_{2v1}$. We first describe the sub-team decomposition in the next section, which also serves as a baseline for our proposed hybrid approach.

3 Adaptive Sub-Team Decomposition

In this section, we formalize a decomposition-based approach that utilizes primitive sub-team games to approximate the multi-agent interaction of the original game. We describe the approach from the Red team’s perspective, which acts as the maximizer.

A key observation underlying the decomposition approach is that the offline-computed primitive sub-team game value functions quantify the equilibrium outcome that can be expected from the corresponding local interactions. Therefore, by partitioning the multi-agent game into a collection of primitive sub-team games and aggregating their values, we can obtain a surrogate measure of the evaluation of the joint state. This surrogate can then be used to identify a favorable assignment of agents to primitive interactions and induce a localized sub-team game policy. To formalize the proposed decomposition approach, let $s_t = (s_t^r, s_t^b)$ denote the joint state of the game at time t . Given $\mathcal{R} \subseteq \mathcal{N}_r$ and $\mathcal{B} \subseteq \mathcal{N}_b$, We first define a sub-team game as a tuple $g = \langle (\mathcal{R}, \mathcal{B}), G, V_e, \mathcal{P}, \mathcal{R}, \Gamma^G, \beta \rangle$, isolating a subset of Red agents $\mathcal{R}$ and Blue agents $\mathcal{B}$ into a localized interaction while inheriting the underlying structure from the original game $\mathcal{G}$. Each sub-team game g corresponds to one of the available primitive types, meaning the team sizes are restricted to $(|\mathcal{R}|, |\mathcal{B}|) \in \{(1, 1), (2, 1)\}$ for our illustrative case. For a given sub-team game g , the local state is defined as $s_t^g = (s_t^{\mathcal{R}}, s_t^{\mathcal{B}})$, where $s_t^{\mathcal{R}}$ and $s_t^{\mathcal{B}}$ denote the states of the agents present in $\mathcal{R}$ and $\mathcal{B}$, respectively.

We define a decomposition $d(s_t)$ of the joint state s_t at time t as a collection of sub-team games, i.e., $d(s_t) = \{g_1, g_2, \dots, g_L\}$, where L denotes the total number of sub-team games. Note that $g_1, g_2, \dots, g_L$ depend on t , but we drop this dependence for notational convenience. The set of admissible decompositions, denoted by $\mathcal{D}$, consists of all such collections satisfying the partition constraints

$$\bigcup_{\ell=1}^L \mathcal{R}_\ell = \mathcal{N}_r, \quad \bigcup_{\ell=1}^L \mathcal{B}_\ell = \mathcal{N}_b, \quad \text{with} \quad \mathcal{R}_\ell \cap \mathcal{R}_{\ell'} = \mathcal{B}_\ell \cap \mathcal{B}_{\ell'} = \emptyset \quad \forall \ell \neq \ell'. \quad (6)$$

That is, each agent is assigned to exactly one sub-team game, and no agent participates in more than one sub-team game. For example, consider a configuration with three Red agents and two Blue agents. Utilizing the 1v1 and 2v1 primitive sub-team game, there are six valid decompositions as illustrated in Fig. 2.

Each decomposition induces a distinct set of local interactions and can lead to different outcomes; the goal is therefore to identify the assignment that best

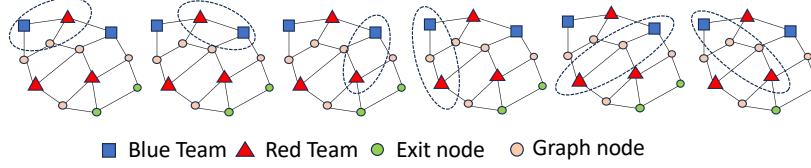


Fig. 2: Illustration of all six valid decompositions for a configuration with 3 Red agents and 2 Blue agents into primitive sub-team games on a random graph with two goal nodes. Each decomposition consists of one 1v1 sub-team game (indicated by the dotted ellipsoid) and one 2v1 sub-team game, formed by pairing the remaining Blue agent with the other two Red agents.

approximates the outcome of the full game. Given a decomposition $d \in \mathcal{D}$, we use a surrogate value function to evaluate the decomposition at s_t by summing the pre-computed value functions of its constituent sub-team games, i.e.,

$$\bar{V}(d, s_t) = \sum_{g_\ell \in d(s_t)} V_{g_\ell}^*(s_t^{g_\ell}), \quad (7)$$

where $V_{g_\ell}^*$ is the corresponding primitive sub-team game value function (V_{1v1}^* or V_{2v1}^* based on the team sizes $|\mathcal{R}_\ell|, |\mathcal{B}_\ell|$). The optimal decomposition is the partition that maximizes this surrogate value:

$$d^*(s_t) = \arg \max_{d \in \mathcal{D}} \bar{V}(d, s_t). \quad (8)$$

Once the optimal decomposition $d^*(s_t)$ is computed, it induces a decentralized *sub-team policy*. Under this policy, action selection is executed at the sub-team game level. For each assigned sub-team game $g_\ell \in d^*(s_t)$, the subset of Red team agents $\mathcal{R}_\ell$ collectively observes the local joint state $s_t^{g_\ell}$ and samples a joint sub-team action $a^{\mathcal{R}_\ell} \sim \pi_{g_\ell}^{r*}(s_t^{g_\ell})$ according to the pre-computed primitive sub-team Nash policy (π_{1v1}^{r*} or π_{2v1}^{r*} based on the team sizes $|\mathcal{R}_\ell|, |\mathcal{B}_\ell|$). Each individual agent $i \in \mathcal{R}_\ell$ then executes its corresponding local action a^i derived from the sampled joint action $a^{\mathcal{R}_\ell}$. This decomposition, evaluation, and assignment process is executed dynamically at every time step throughout the game's execution. However, as discussed previously, a decomposition-based approach has limited ability to capture coordination at the team level. This limitation motivates the tree-search algorithm presented in the next section, which explicitly accounts for coordination among agents at the team level by sampling joint actions via rollouts.

4 Primitive-Guided Tree Search

In this section, we present PGTS, a hybrid online planning framework for multi-agent games that combines offline-computed primitive sub-team game solutions with online tree search, building on the Simultaneous-Move Monte Carlo Tree Search (SM-MCTS) framework [20]. As discussed already, directly applying tree search in the joint action space is computationally infeasible due to the exponential branching factor. Our approach addresses this challenge by incorporating

structure from the primitive sub-team games into both expansion and leaf node value estimation.

We describe the algorithm from the Red team’s perspective (maximizer); the tree is built in exactly the same way for the Blue team (minimizer), with the only difference being the selection and backpropagation steps. At each execution time step t , the current joint state $s_t = (s_t^r, s_t^b)$ initializes the root node of the tree $\mathcal{T}$. The tree is built incrementally through M repeated simulations where M denotes the total simulation budget before a final action is executed in the environment. Each simulation executes the following phases (Fig. 1):

1. **Selection:** The tree is traversed by selecting joint actions until a terminal or unexpanded node is reached.
2. **Expansion:** The unexpanded node is expanded to generate new child nodes.
3. **Leaf Value Evaluation:** The newly generated nodes are evaluated to estimate the future game outcome.
4. **Backpropagation:** The resulting reward or surrogate estimate is passed back up the visited trajectory to update the node statistics.

Since exhaustive search is computationally infeasible in large multi-agent games, planning is performed over a finite horizon by limiting the maximum search depth to D . Our primary contributions lie in designing the structured **Expansion** phase and a scalable **Leaf Value Evaluation** mechanism tailored to the multi-agent setting. The pseudo-code for the overall approach is provided in Algorithm 3.

4.1 Primitive-Guided Expansion

A key challenge in multi-agent tree search is to identify promising actions without exhaustively expanding the combinatorially large joint action space. While heuristic action-sampling schemes can reduce the branching factor, they often fail to exploit the strategic structure of the underlying game. To address this challenge, we perform a *primitive-guided expansion* to guide expansion toward actions that are likely to be relevant under strategic play, thereby focusing the search budget on strategically promising regions of the action space.

During the expansion of a node n in the search tree $\mathcal{T}$, let $s_n = (s_n^r, s_n^b)$ denote the joint state associated with node n . Each Blue agent $j \in \mathcal{N}_b$ identifies its k_r nearest Red opponents, denoted by the neighborhood set $\mathcal{N}_{k_r}(j) \subseteq \mathcal{N}_r$. We then sample localized interactions using the pre-computed primitive sub-team game Nash policies:

- **1v1 samples:** $(a^{r,i}, a^{b,j}) \sim \pi_{1v1}^*(s_n^i, s_n^j)$ for $i \in \mathcal{N}_{k_r}(j)$. Specifically, for each Blue agent j , we enumerate individual Red agents in its neighborhood set $\mathcal{N}_{k_r}(j)$. For each pair of a Red agent and a Blue agent, we sample their joint actions according to π_{1v1}^* based on their current joint state.
- **2v1 samples:** $(a^{r,i_1}, a^{r,i_2}, a^{b,j}) \sim \pi_{2v1}^*((s_n^{i_1}, s_n^{i_2}), s_n^j)$ for unordered pairs $\{i_1, i_2\} \subseteq \mathcal{N}_{k_r}(j)$. Similar to the 1v1 case, we enumerate pairs of two Red agents both in Blue agent j ’s neighborhood $\mathcal{N}_{k_r}(j)$. For each two-Red-one-Blue group, we sample the three agents’ actions from π_{2v1}^* .

We aggregate these local action samples across all Blue agent neighborhoods to construct a restricted candidate action set for the Red team $\tilde{\mathcal{A}}^r(s_n^r) \subseteq \mathcal{A}^r(s_n^r)$ and the Blue team $\tilde{\mathcal{A}}^b(s_n^b) \subseteq \mathcal{A}^b(s_n^b)$. The restricted joint action space evaluated at node n is therefore: $\tilde{\mathcal{A}}(s_n) = \tilde{\mathcal{A}}^r(s_n^r) \times \tilde{\mathcal{A}}^b(s_n^b) \subseteq \mathcal{A}(s_n)$. To maintain exploration and prevent the search from overcommitting to the primitive priors, we augment the restricted action set with heuristic shortest-path actions toward relevant opponents/exits, as well as randomly sampled joint team actions. Node n is then expanded by instantiating a new child node n' for each joint action $a = (a^r, a^b) \in \tilde{\mathcal{A}}(s_n)$. This structured expansion mechanism reduces the branching factor while biasing the search toward strategically relevant actions. Additionally, to maintain tractability, agents that are captured or have escaped upon reaching the new state $s_{n'}$ are removed based on the game rules Γ^G , and only the remaining agents are simulated thereafter.

4.2 Leaf Value Estimation via Sub-team Surrogates

The tree-search algorithm requires an estimate of the value for each leaf node in the search tree $\mathcal{T}$. Rather than performing expensive Monte-Carlo rollouts or learning a separate value function, we approximate the value of a leaf node using the decomposition framework introduced in Section 3, which leverages the pre-computed value functions of the *primitive sub-team games*.

Let s denote the joint state, getting an estimate of the value of state s requires solving $\max_{d \in \mathcal{D}} \bar{V}(d, s)$ over the set of admissible decompositions $\mathcal{D}$. However, finding the optimal decomposition scales combinatorially with the number of agents. To obtain a scalable approximation suitable for online planning, we relax this optimization problem to a structured two-stage sequential matching problem.

Crucially, we prioritize extracting the 2v1 interactions first. Higher-order primitives capture essential multi-agent coordination such as joint trapping that isolated 1v1 pairings fundamentally ignore. By matching the 2v1 sub-team interactions before the 1v1 ones, we ensure that the leaf-value estimate reflects the highest degree of team-level synergy available before falling back to independent 1v1 pursuits. To operationalize the sequential matching process, let $\mathcal{N}_r^{\text{rem}} \subseteq \mathcal{N}_r$ and $\mathcal{N}_b^{\text{rem}} \subseteq \mathcal{N}_b$ denote the set of Red and Blue team agents that remain in the game at node n' , with $|\mathcal{N}_r^{\text{rem}}| = n_r$ and $|\mathcal{N}_b^{\text{rem}}| = n_b$. We first determine the target number of 2v1 and 1v1 sub-team interactions (n_{2v1} and n_{1v1} , respectively) to be considered by checking the partition constraints

$$n_r = 2n_{2v1} + n_{1v1}, \quad n_b = n_{2v1} + n_{1v1}.$$

If an exact decomposition is not feasible for a given team configuration, we greedily maximize the number of feasible 2v1 interactions whenever the Red team has numerical superiority; otherwise, only 1v1 interactions are considered. We first construct the candidate 2v1 interactions by defining the set of unordered Red pairs $\mathcal{P}_{\mathcal{N}_r^{\text{rem}}} = \{\{i, j\} \subseteq \mathcal{N}_r^{\text{rem}} \mid i \neq j\}$. For every pair $\{i, j\} \in \mathcal{P}_{\mathcal{N}_r^{\text{rem}}}$ and $k \in \mathcal{N}_b^{\text{rem}}$, we populate the 2v1 cost matrix using the 2v1 primitive sub-team game

value function : $C_{2v1}(\{i, j\}, k) := V_{2v1}^*((s^i, s^j), s^k)$. We then solve a maximum-sum linear assignment over $(\mathcal{P}_{\mathcal{N}_r^{\text{rem}}}, \mathcal{N}_b^{\text{rem}})$ using the Hungarian algorithm [12], yielding a matching $M_{2v1} \subseteq \mathcal{P}_{\mathcal{N}_r^{\text{rem}}} \times \mathcal{N}_b^{\text{rem}}$ ¹. The top n_{2v1} highest-value matched tuples are selected, and their constituent agents are removed from the remaining-agent sets $\mathcal{N}_r^{\text{rem}}$ and $\mathcal{N}_b^{\text{rem}}$. For the agents that remain unmatched after the 2v1 matching, we define a 1v1 cost matrix : $C_{1v1}(i, j) := V_{1v1}^*(s^i, s^j)$, and solve a second linear assignment problem to obtain the 1v1 matching $M_{1v1} \subseteq \mathcal{N}_r^{\text{rem}} \times \mathcal{N}_b^{\text{rem}}$. Finally, the leaf value estimate $V^{\text{leaf}}(s)$ is computed as the total aggregated sum of the selected primitive sub-games:

$$V^{\text{leaf}}(s) = \sum_{(\{i, j\}, k) \in M_{2v1}} V_{2v1}^*((s^i, s^j), s^k) + \sum_{(i, j) \in M_{1v1}} V_{1v1}^*(s^i, s^j). \quad (9)$$

4.3 Action Selection and Update

During the selection phase of each simulation $m \in \{1, 2, \dots, M\}$, the algorithm recursively traverses the tree $\mathcal{T}$ by selecting joint actions until a terminal state, an unexpanded node, or the maximum search depth D is reached. When an unexpanded node is encountered, it is expanded, and its child nodes are evaluated using the value estimate described in the previous section. The resulting value estimate is then backpropagated along the visited trajectory to update the statistics maintained by the selection operator. As before, let n denote a node in the tree $\mathcal{T}$ visited during this traversal, and let $s_n = (s_n^r, s_n^b)$ denote the joint state associated with that node. Action selection follows the standard SM-MCTS framework [20] and is treated as a modular component. At each visited node n , each team $* \in \{r, b\}$ selects an action $a^* \in \tilde{\mathcal{A}}^*(s_n^*)$ according to its selection rule. The resulting joint action pair $a = (a^r, a^b)$ determines the successor node in the search tree. We consider two commonly used action selection operators: Regret matching (RM) and Decoupled Upper Confidence bounds applied to Trees (DUCT). These are described next.

Regret Matching (RM): This selection operator applies regret matching [10] to the estimated matrix game at every node n in the tree $\mathcal{T}$ to compute a mixed policy for both teams. Let $\hat{Q}(s_n) \in \mathbb{R}^{|\tilde{\mathcal{A}}^r(s_n^r)| \times |\tilde{\mathcal{A}}^b(s_n^b)|}$ denote the *empirical* action-value matrix estimated dynamically during the tree search, and let $\hat{Q}(s_n, a^r, a^b)$ represent the estimated value for each joint action $(a^r, a^b) \in \tilde{\mathcal{A}}^r(s_n^r) \times \tilde{\mathcal{A}}^b(s_n^b)$ selected $N(s_n, a^r, a^b)$ times. The entries of $\hat{Q}(s_n)$ are initialized using the simulated expansion return (see Algorithm 3 **ExpandNode** function) to ensure that all joint actions have non-zero initial values. Additionally, each team $* \in \{r, b\}$ maintains cumulative regrets $r^*(s_n, a^*)$ for not playing joint action a^* at state s_n , and an average policy $\bar{\sigma}^*(s_n, a^*)$, which accumulates the selection policies over all visits. All tracking variables are initialized to 0. The selection and update procedures are summarized in Algorithm 1.

¹ Strictly enforcing the disjointness constraint requires solving the combinatorial problem. To enable efficient online planning, we relax this constraint during the 2v1 matching stage, allowing a Red agent to appear in more than one matched pair.

Algorithm 1 Regret (RM) Selection and Update

```

1: Let  $s_n$  be the joint state associated with node  $n$ , and  $\phi_r = +1$ ,  $\phi_b = -1$ 
2: function INITIALIZENODESTATS( $\mathcal{T}, n, (a^r, a^b), Q^{\text{init}}$ )
3:    $\hat{Q}(s_n, a^r, a^b) \leftarrow Q^{\text{init}}$ ,  $N(s_n, a^r, a^b) \leftarrow 1$   $\triangleright$  Initialize  $\hat{Q}$  estimates and visits
4:   for all team  $*$   $\in \{r, b\}$  do  $\triangleright$  Initialize regrets and average policy
5:      $r^*(s_n, a^*) \leftarrow 0$ ,  $\bar{\sigma}^*(s_n, a^*) \leftarrow 0$ 
6:   function SELECTJOINTACTION( $\mathcal{T}, n, m$ )
7:     for all team  $*$   $\in \{r, b\}$  do
8:       // policy  $\propto$  cumulative positive regret
9:        $\sigma_m^*(s_n, a^*) \leftarrow \frac{[r^*(s_n, a^*)]^+}{\sum_{\tilde{a}^*} [r^*(s_n, \tilde{a}^*)]^+}$  if denominator  $> 0$ , else uniform
10:      Sample  $a^* \sim (1 - \gamma_{rm}) \sigma_m^*(s_n, a^*) + \frac{\gamma_{rm}}{|\tilde{\mathcal{A}}^*(s_n)|}$   $\triangleright \gamma_{rm}$ -greedy exploration
11:    return  $(a^r, a^b)$ 
12:   function UPdatenodeStats( $\mathcal{T}, n, (a^r, a^b), v, m$ )
13:     // Update estimated value matrix
14:      $\hat{Q}(s_n, a^r, a^b) \leftarrow \hat{Q}(s_n, a^r, a^b) + v$ ,  $N(s_n, a^r, a^b) \leftarrow N(s_n, a^r, a^b) + 1$ 
15:     // Update counterfactual regrets based on sampled opponent action
16:     for all team  $*$   $\in \{r, b\}$ ,  $\tilde{a}^* \in \tilde{\mathcal{A}}^*(s_n) \setminus \{a^*\}$  do
17:        $r^*(s_n, \tilde{a}^*) \leftarrow r^*(s_n, \tilde{a}^*) + \phi_* \left( \frac{\hat{Q}(s_n, \tilde{a}^*, a^*)}{N(s_n, \tilde{a}^*, a^*)} - v \right)$ 
18:        $\bar{\sigma}^*(s_n, a^*) \leftarrow \bar{\sigma}^*(s_n, a^*) + \sigma_m^*(s_n, a^*)$   $\triangleright$  Accumulate average policy
19:   function EXTRACTROOTPOLICIES( $\mathcal{T}, n_0$ )
20:     for all team  $*$   $\in \{r, b\}$  do
21:        $\pi^*(a^*) \leftarrow \bar{\sigma}^*(s_{n_0}, a^*) / \sum_{\tilde{a}^*} \bar{\sigma}^*(s_{n_0}, \tilde{a}^*)$   $\triangleright$  Normalize accumulated average
22:   return  $(\pi^r, \pi^b)$ 

```

Decoupled UCT (DUCT): This selection operator decouples the action-selection at every node n in the tree $\mathcal{T}$ by independently applying the Upper Confidence Bound (UCB) criterion to each team's marginal action-value estimates, thereby ignoring the joint coupling during selection. For each node n , team maintains marginal action-value estimates $\hat{Q}^*(s_n, a^*)$ and visit counts $N^*(s_n, a^*)$ for every action $a^* \in \tilde{\mathcal{A}}^*(s_n)$, along with the total node visit count $N(s_n)$. The marginal action values are obtained by aggregating the joint-action value matrix $\hat{Q}(s_n)$ over the opponent's actions. Similar to RM, these marginal values are initialized using the expansion return. The selection and update procedures are summarized in Algorithm 2.

Algorithm 2 Decoupled UCT (DUCT) Selection and Update

```

1: Let  $s_n$  be the joint state associated with node  $n$ ,  $\phi_r = +1$ ,  $\phi_b = -1$ 
2: function INITIALIZENODESTATS( $\mathcal{T}, n, (a^r, a^b), Q^{\text{init}}$ )
3:    $N(s_n) \leftarrow 1$ 
4:   for all team  $*$   $\in \{r, b\}$  do
5:      $\hat{Q}^*(s_n, a^*) \leftarrow Q^{\text{init}}$ ,  $N^*(s_n, a^*) \leftarrow 1$ 
6:   function UPdatenodeStats( $\mathcal{T}, n, (a^r, a^b), v, m$ )
7:      $N(s_n) \leftarrow N(s_n) + 1$ 
8:     for all team  $*$   $\in \{r, b\}$  do  $\triangleright$  Update marginal statistics independently
9:        $\hat{Q}^*(s_n, a^*) \leftarrow \hat{Q}^*(s_n, a^*) + v$ ,  $N^*(s_n, a^*) \leftarrow N^*(s_n, a^*) + 1$ 

```

```

10: function SELECTJOINTACTION( $\mathcal{T}, n, m$ )
11:   // Red Team (Maximizer) uses UCB, Blue Team (Minimizer) uses LCB
12:   for all team  $*$   $\in \{r, b\}$  do
13:      $a^* \leftarrow \arg \max_{a^* \in \tilde{\mathcal{A}}^*(s_n^*)} \left\{ \phi_* \frac{\hat{Q}^*(s_n, a^*)}{N^*(s_n, a^*)} + C \sqrt{\frac{\ln(N(s_n))}{N^*(s_n, a^*)}} \right\}$ 
14:   return  $(a^r, a^b)$ 
15: function EXTRACTROOTPOLICIES( $\mathcal{T}, n_0$ )
16:   for all team  $*$   $\in \{r, b\}$  do
17:      $\pi^* \leftarrow \arg \max_{a^*} \phi_* \frac{\hat{Q}^*(s_{n_0}, a^*)}{N^*(s_{n_0}, a^*)}$   $\triangleright$  Greedy w.r.t. mean reward
18:   return  $(\pi^r, \pi^b)$ 

```

Algorithm 3 Primitive-Guided Tree Search

```

1: Require: Joint state  $s_t$ , simulations  $M$ , max depth  $D$ , game rules  $\Gamma^G$ , neighbor
   param  $k_r$ , discount  $\beta$ 
2: Offline: Solve primitives once before game begins  $\pi_{1v1}^*, V_{1v1}^*, \pi_{2v1}^*, V_{2v1}^*$ 
3: Online: Construct tree  $\mathcal{T}$  with root node  $n_0$  corresponding to  $s_t$ 
4: for simulation  $m = 1$  to  $M$  do  $\triangleright$  Execute  $M$  simulations
5:   SIMULATE( $\mathcal{T}, n_0, m, 0$ )  $\triangleright$  Start simulation at depth  $d = 0$ 
6:    $(\pi^r, \pi^b) \leftarrow$  EXTRACTROOTPOLICIES( $\mathcal{T}, n_0$ )  $\triangleright$  via RM(Alg. 1) or DUCT(Alg. 2)
7:   function SIMULATE( $\mathcal{T}, n, m, d$ )
8:     Let  $s_n$  be the joint state of the agents associated with node  $n$ 
9:     if  $s_n$  is empty then return 0  $\triangleright$  Terminal : No active agent remaining
10:    else if  $n$  is not expanded in  $\mathcal{T}$  then
11:      EXPANDNODE( $\mathcal{T}, n$ )
12:       $(a^r, a^b) \leftarrow$  SELECTJOINTACTION( $\mathcal{T}, n, m$ )  $\triangleright$  via RM or DUCT
13:       $Q^{\text{est}} \leftarrow$  GETPRIORVALUE( $n, (a^r, a^b)$ )  $\triangleright$  set during EXPANDNODE
14:      UPDATENODESTATS( $\mathcal{T}, n, (a^r, a^b), Q^{\text{est}}, m$ )  $\triangleright$  via RM or DUCT
15:      return  $Q^{\text{est}}$ 
16:    else  $\triangleright$  Internal node: select joint action and recurse
17:       $(a^r, a^b) \leftarrow$  SELECTJOINTACTION( $\mathcal{T}, n, m$ )  $\triangleright$  via RM or DUCT
18:       $n' \leftarrow$  child of  $n$  after selecting joint action
19:      if  $d + 1 = D$  then  $\triangleright$  Search limit reached; bypass recursion
20:         $Q^{\text{est}} \leftarrow$  GETPRIORVALUE( $n, (a^r, a^b)$ )  $\triangleright$  set during EXPANDNODE
21:      else  $\triangleright$  Recursive descent
22:         $v_{\text{child}} \leftarrow$  SIMULATE( $\mathcal{T}, n', m, d + 1$ )
23:         $Q^{\text{est}} \leftarrow \mathcal{R}(s, a^r, a^b) + \beta v_{\text{child}}$   $\triangleright$  Query the immediate reward
24:      // Backpropagate statistics
25:      UPDATENODESTATS( $\mathcal{T}, n, (a^r, a^b), Q^{\text{est}}, m$ )  $\triangleright$  via RM or DUCT
26:      return  $Q^{\text{est}}$ 
27:   end function
28: function EXPANDNODE( $\mathcal{T}, n$ )
29:   Let  $s_n$  be the joint state of the agents associated with node  $n$ 
30:    $\tilde{\mathcal{A}}(s_n) \leftarrow$  PRIMITIVEGUIDEDEXPANSION( $s_n, \pi_{1v1}^*, \pi_{2v1}^*, k_r$ )
31:   for all joint actions  $(a^r, a^b) \in \tilde{\mathcal{A}}(s_n)$  do
32:     Add child node  $n'$  to  $\mathcal{T}$ 
33:      $s_{n'}, \mathcal{R}(s_n, a^r, a^b) \leftarrow$  APPLYGAMERULES( $\mathcal{P}(s_n, (a^r, a^b)), \Gamma^G$ )
34:     // Compute value estimate for the newly added node
35:      $V^{\text{leaf}}(s_{n'}) \leftarrow$  LEAFVALUEESTIMATION( $s_{n'}, V_{1v1}^*, V_{2v1}^*$ )  $\triangleright$  Eq. (9)
36:      $Q^{\text{init}}(s_n, a^r, a^b) \leftarrow \mathcal{R}(s_n, a^r, a^b) + \beta V^{\text{leaf}}(s_{n'})$ 
37:     INITIALIZENODESTATS( $n, (a^r, a^b), Q^{\text{init}}$ )  $\triangleright$  Warm-start priors for selection
38:   end function

```

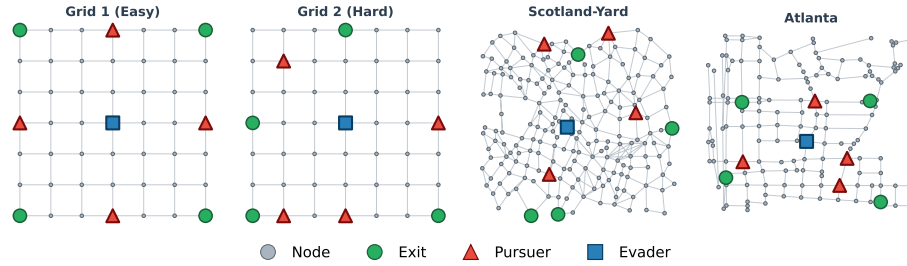


Fig. 3: Experimental environments used in evaluation.

5 Experiments

In this section, we evaluate the performance of the primitive-guided tree search PGTS for a graph-based multi-agent Pursuit-Evasion game, in which a team of Pursuers (Red) is tasked with capturing a team of Evaders (Blue) before they can reach the exit nodes (Green) and escape (Fig. 3).

5.1 Setup

To provide a rigorous benchmark, we utilize the open-source *GraphChase* platform [24], which contains state-of-the-art (SOTA) algorithms for graph-based PEGs. While the existing methods on this platform are constrained to N -Pursuer versus 1-Evader $Nv1$ scenarios, our proposed tree search is generalizable to arbitrary multi-agent configuration NvM . For a direct and fair comparison with the SOTA baselines, we mainly focus on analyzing the $Nv1$ setting.

Game rules Γ^G : The capture and escape radius are both zero. That is, an Evader is captured when a Pursuer occupies the same node, and successfully evades when it reaches an Evasion node. The game terminates upon the Evader’s capture or escape, or when the maximum time horizon T is reached². If the Evader is neither captured nor has escaped by the end of the time horizon, the Pursuer team receives a reward of +1 for successfully defending.

Evaluation maps: We performed experiments on three graph environments natively implemented in GraphChase [24]: two 7×7 grid graphs and a 200-node Scotland Yard graph from GraphChase. We also added a real-world graph of Atlanta with 151 nodes. Details on the graphs and agent configurations are provided in Fig. 3.

Evaluation Metrics: Following the convention established in *GraphChase*, we evaluate the Pursuer team’s performance using the following metrics:

- **Worst-case utility (WCU):** Given the initial joint state $s_0 = (s_0^p, s_0^e)$, let $\mathcal{T}_e^T(s_0^e)$ denote the set of all feasible Evader trajectories from the initial position s_0^e leading to any exit node within the time horizon T . For a given Pursuer

² The time horizon of the game is finite; for simplicity, we solve for the primitive-sub-team games by assuming the time horizon is infinity.

team policy π_p , the worst-case utility is defined as

$$U_{\text{worst}}(\pi_p \mid s_0, \tau_e) = \min_{\tau_e \in \mathcal{T}_e^T(s_0^e)} \mathbb{E}_{\pi_p} \left[\sum_{t=0}^T \mathcal{R}(s_t, a_t^p, a_t^e) \mid s_0, \tau_e \right], \quad (10)$$

where the expectation is taken over the Pursuer team’s actions induced by policy π_p , while the Evader’s action a_t^e deterministically follows the given trajectory τ_e . This metric evaluates the robustness of a Pursuer team’s policy against different Evader behaviors.

- **Worst-case utility against shortest-path Evader (SP-WCU):** To facilitate a fair comparison with the learning-based baselines in GraphChase, which are trained against shortest-path Evaders, we additionally evaluate the worst-case utility over the set of shortest paths connecting the Evader’s initial position to each exit node to ensure the training has been completed.
- **Expected reward (ER):** The expected game reward obtained under the interaction of a particular Pursuer and Evader team policy.

Baselines: We compared PGTS against four benchmarking algorithms spanning learning-based, decomposition-based, and heuristic-based Pursuer policies.

- (i) **Multi-Task PSRO (MT-PSRO) [13]:** A state-of-the-art PSRO-based framework that solves multi-agent games by iteratively adding new best-response policies to a growing meta-game. It uses Multi-Agent Proximal Policy Optimization (MAPPO) [23] as the best-response oracle for the Pursuer team and uses shortest path Evader in the training pipeline. MT-PSRO is natively implemented in the GraphChase package.
- (ii) **NSGZero [22]:** A neural-network-guided Monte-Carlo Tree Search (MCTS) framework developed for network security games. NSGZero is natively implemented in the GraphChase package.
- (iii) **Decomposition³:** The sub-team decomposition policy from Section 3.
- (iv) **Intercepting:** A heuristic rule-based baseline where Pursuers execute a greedy shortest-path policy to intercept the Evader based on its projected path to the nearest exit node.

5.2 Results and Discussions

We present a comprehensive analysis of the experimental results across different graph topologies. Table 1 compares the Pursuer team policy in terms of worst-case utility (WCU), shortest-path worst-case utility (SP-WCU), and expected reward (ER). The expected reward is evaluated over Evader trajectories in $\mathcal{T}_e^T$. To account for stochasticity in the policies, all reported metrics are averaged over 400 repeated runs for the grid graphs and 200 runs for the Scotland-Yard and Atlanta graphs. The tree-search variants use Regret Matching (RM) or Decoupled UCT (DUCT) during action selection⁴.

³ To decompose the Nv1 game into primitive subgames while including all Pursuers, each subgame is defined against the same Evader

⁴ To ensure a fair comparison, the hyperparameters for both variants are tuned against the shortest-path Evader and held fixed during the worst-case utility evaluation. Additional details are provided in Appendix A.

		Intercepting	Decomp.	MT-PSRO	NSGZero	PGTS-RM	PGTS-DUCT
Grid 1 (easy)	WCU	0.43	0.97	0.94	0.96	0.99	0.99
	ER	0.60	1.0	0.97	1.0	1.0	1.0
Grid 2 (hard)	SP-WCU	0.09	0.09	0.47	0.45	0.46	0.40
	WCU	0.09	0.06	0.25	0.37	0.46	0.40
	ER	0.78	0.82	0.70	0.80	0.90	0.91
Scotland-Yard	SP-WCU	0.07	0.33	0.97	0.98	0.82	0.96
	WCU	0.00	0.27	0.05	0.00	0.73	0.68
	ER	0.69	0.82	0.49	0.54	0.98	0.99
Atlanta	SP-WCU	0.20	0.50	0.91	1.0	0.89	0.94
	WCU	0.00	0.47	0.49	0.00	0.87	0.94
	ER	0.80	0.92	0.91	0.94	1.0	1.0

Table 1: Comparison of the worst-case utility (WCU), shortest-path worst-case utility (SP-WCU), and expected reward (ER) achieved by the Pursuer team across different graph topologies. WCU is evaluated against all admissible Evader trajectories in $\mathcal{T}_e^T$, SP-WCU against a shortest-path Evader, and ER against Evader trajectories sampled from $\mathcal{T}_e^T$. The game horizon is $T=6$ for the grid environments and $T=9$ for the Scotland Yard and Atlanta environments.

Both the proposed PGTS-RM and PGTS-DUCT consistently achieve strong performance across all graph topologies. While Grid 1 serves as a sanity check in which most methods perform well, the performance gap widens as the game complexity increases. On Grid 2, the tree-search methods maintain high utility under both evaluations, exhibiting clear advantages over the baseline approaches. The benefit of online search is most apparent for the Scotland-Yard and Atlanta graphs, where the tree-search variants worst-case utility is substantially higher than the baselines.

The sub-team decomposition framework also provides a valuable strategic prior. The Decomposition Pursuer remains competitive with learned baselines against a strategic Evader. However, decomposition alone is much less effective than tree search across all graph topologies, underscoring the importance of team coordination. This limitation arises because the decomposition-based approach focuses on agent coordination within each primitive sub-team game, but ignores joint coordination among agents at the team level. To further evaluate performance, we compared the various Pursuer team policies against PGTS Evader in Table 2. Our PGTS Pursuers consistently achieve the highest rewards.

In addition, although MT-PSRO and NSGZero achieve stronger performance against shortest-path Evaders, their policies appear to be overfitted to the shortest-path Evaders (SP-WCU) used during training. As a result, they struggle against more complex or unseen Evader behaviors, leading to substantially lower worst-case utility (Table 1), as well as weaker performance against PGTS-based Evaders (Table 2). In contrast, the PGTS Pursuer maintains robust performance against all variants of Evader policies since it evaluates the game tree dynamically, rather than relying solely on precomputed historical data distributions.

5.3 Ablation Study

To better understand the contribution of the individual components of the proposed PGTS framework, we performed an ablation study on both the Regret

Evader policies	Pursuer policies		Intercepting Decomp. MT-PSRO NSGZero PGTS-RM PGTS-DUCT					
Grid 1 (easy)	PGTS-RM	T=6	0.61	1.0	0.99	1.0	1.0	1.0
		T=12	0.10	1.0	0.62	0.14	0.98	0.98
	PGTS-DUCT	T=6	0.50	1.0	0.98	1.0	1.0	1.0
		T=12	0.08	1.0	0.62	0.06	0.98	1.0
Grid 2 (hard)	PGTS-RM	T=6	0.09	0.19	0.03	0.04	0.23	0.27
		T=12	0.02	0.15	0.01	0.02	0.23	0.27
	PGTS-DUCT	T=6	0.03	0.15	0.02	0.07	0.07	0.11
		T=12	0.02	0.11	0.01	0.04	0.07	0.10
Scotland-Yard	PGTS-RM	T=9	0.02	0.93	0.76	0.77	0.98	0.99
		T=20	0.01	0.80	0.14	0.18	0.87	0.95
	PGTS-DUCT	T=9	0.0	0.94	0.72	0.13	0.98	0.99
		T=20	0.0	0.72	0.18	0.03	0.94	0.96
Atlanta	PGTS-RM	T=9	0.21	0.87	0.87	1.0	1.0	1.0
		T=20	0.15	0.71	0.30	0.31	0.97	0.99
	PGTS-DUCT	T=9	0.20	0.82	0.97	1.0	1.0	1.0
		T=20	0.14	0.87	0.31	0.25	0.93	1.0

Table 2: Comparison of the expected Pursuer team reward against tree search Evader across two time horizons T . PGTS-RM and PGTS-DUCT denote the proposed tree search with Regret Matching and Decoupled UCT selection operators, respectively.

	Grid 1 (easy)	Grid 2 (hard)	Scotland-Yard	Atlanta
PGTS-RM	0.99	0.46	0.73	0.89
TS-RM-wo-PG-Expansion	0.89	0.27	0.58	0.73
PGTS-RM-1v1	0.65	0.47	0.15	0.68
PGTS-DUCT	0.99	0.40	0.68	0.94
TS-DUCT-wo-PG-Expansion	0.95	0.22	0.30	0.90
PGTS-DUCT-1v1	0.81	0.31	0.12	0.69

Table 3: Comparison of the worst-case utility (WCU) for the variants of the tree search Pursuer team policy across different graphs. The results isolate the impact of primitive-guided expansion and 2v1 primitive inclusion.

Matching (RM) and Decoupled UCT (DUCT) variants. Specifically, we evaluated the importance of (i) primitive-guided expansion to focus the search on strategically relevant actions, and (ii) higher-order 2v1 primitive interactions to capture coordinated team behavior. We compared the full PGTS against two degraded variants.

1. **Tree search w/o Primitive Guided Expansion:** Primitive Nash sub-team game policies are removed from the expansion and rollout phases and replaced with hand-crafted heuristic policies. The Evader selects shortest-path actions toward all the exit nodes, while the Pursuers select shortest-path actions toward the Evader, the nearest exit node, and an interception point. The full decomposition-based leaf-value estimator, including both 1v1 and 2v1 primitives, is retained.
2. **PGTS with only one primitive 1v1:** The tree-search algorithm with only the 1v1 primitive subgame. Specifically, the higher-order 2v1 primitive is omitted from both the guided expansion phase and the leaf-value estimator.

The results of the ablation study are summarized in Table 3. Overall, both primitive-guided expansion and higher-order 2v1 primitives contribute meaningfully to the performance of PGTS. While primitive-guided expansion improves the quality of exploration, the inclusion of 2v1 primitive allows the search to reason about coordinated team interactions beyond independent 1v1 pursuits. For the Grid 2 configuration, the performance gap is relatively small, because

the chosen initial configuration leads to game termination within a few steps and thus does not require deep lookahead. The benefits of both components become increasingly pronounced on larger graph instances, such as Scotland-Yard and Atlanta, where the longer game horizon makes effective exploration and accurate value estimation particularly important.

6 Conclusion and Future Work

We have presented Primitive-Guided Tree Search (PGTS), a hybrid framework that integrates offline Nash computation with online tree search. By solving a collection of small primitive sub-team games offline and embedding their policies and value functions into the search process, PGTS balances computational tractability with coordinated team-level decision making. PGTS achieves robust performance across diverse graph topologies against state-of-the-art methods.

Future work will focus on extending the proposed framework beyond pursuit-evasion settings to a broader class of multi-agent games. We also plan to evaluate the approach across a wider range of team sizes and agent configurations to better understand its scalability and robustness. Finally, the primitive-guided tree-search framework is largely agnostic to the underlying information structure of the game. It is therefore worth investigating extensions to settings with partial observability and asymmetric information, thus enabling the approach to address a broader range of realistic multi-agent settings.

Acknowledgements: This research was supported by the U.S. Army Research Laboratory through the Distributed and Collaborative Intelligent Systems and Technology (DCIST) Collaborative Research Alliance under Cooperative Agreement No. W911NF-17-2-0181.

Disclosure of Interests: The authors have no competing interests to declare that are relevant to the content of this article.

A Appendix

All experiments were performed on a workstation equipped with a 24-core 5.7GHz Intel Core Ultra 9 285K CPU and an NVIDIA RTX 5090 GPU. For the offline computation of the primitive sub-team game solutions, we utilized the Nash-equilibrium solver developed in [9]. Since the proposed framework relies on pre-computed primitive policies and value functions, the computational cost of solving these primitives is an important consideration. Table 4 reports both the order of the corresponding state-space sizes and the offline computation times for the 1v1 and 2v1 primitive sub-team games across the different graph topologies.

For online planning, we limit the tree-search horizon to a maximum depth of four and perform at most 2,000 simulations at each decision step. Among the four phases of the tree-search procedure, the expansion phase is the most computationally expensive, as it requires generating candidate actions and estimating value. To improve efficiency, these computations are vectorized, resulting in an average decision time of approximately 0.5 seconds per planning step. The

	No. of Nodes	S_{1v1}	S_{2v1}	S	1v1 Primitive	2v1 Primitive
Grid 1 (easy)	49	2.4×10^3	6.0×10^4	1.3×10^7	0.9 s	56.0 s
Grid 2 (hard)	49	2.4×10^3	6.0×10^4	1.3×10^7	0.9 s	65.3 s
Atlanta	151	2.3×10^4	1.7×10^6	3.4×10^9	22.1 s	2948.7 s
Scotland-Yard	200	4.0×10^4	4.0×10^6	1.4×10^{10}	33 s	23500.7 s

Table 4: Order of State-space sizes and offline computation times for the primitive sub-team games across the evaluated graph topologies.

hyperparameters are summarized in Table 5. Across all environments, we use a discount factor of $\beta = 0.95$.

	PGTS-RM	PGTS-DUCT
Grid 1 (easy)	$M = 2000, D = 4, \gamma_{rm} = 0.6, k_r = 4$	$M = 1000, D = 3, C = 2.0, k_r = 4$
Grid 2 (hard)	$M = 2000, D = 2, \gamma_{rm} = 0.6, k_r = 4$	$M = 1000, D = 4, C = 2.0, k_r = 4$
Scotland-Yard	$M = 2000, D = 4, \gamma_{rm} = 0.4, k_r = 3$	$M = 2000, D = 4, C = 2.0, k_r = 3$
Atlanta	$M = 2000, D = 4, \gamma_{rm} = 0.6, k_r = 4$	$M = 2000, D = 4, C = 2.0, k_r = 4$

Table 5: Hyperparameters for the primitive-guided tree search algorithms across different environments. M : number of simulations, D : search depth, γ_{rm} : exploration rate (RM), C : exploration constant (DUCT), k_r : number of neighboring Red agents considered during primitive-guided expansion.

References

1. Becker, T., Sunberg, Z.: Simultaneous AlphaZero: Extending tree search to Markov games (2025), <https://arxiv.org/abs/2512.12486>
2. Bighashdel, A., Wang, Y., McAleer, S., Savani, R., Oliehoek, F.A.: Policy space response oracles: A survey. arXiv preprint arXiv:2403.02227 (2024)
3. Chung, T.H., Hollinger, G.A., Isler, V.: Search and pursuit-evasion in mobile robotics: A survey. *Autonomous robots* **31**(4), 299–316 (2011)
4. Filar, J., Vrieze, K.: *Competitive Markov Decision Processes*. Springer Science & Business Media (2012)
5. Fink, A.M.: Equilibrium in a stochastic n -person game. *Journal of Science of the Hiroshima University, Series A-I (Mathematics)* **28**(1), 89–93 (1964). <https://doi.org/10.32917/hmj/1206139508>
6. Galván-López, E., Li, R., Patsakis, C., Clarke, S., Cahill, V.: Heuristic-based multi-agent Monte Carlo tree search. In: *IISA 2014, The 5th International Conference on Information, Intelligence, Systems and Applications*, Chania, Greece. pp. 177–182 (2014). <https://doi.org/10.1109/IISA.2014.6878747>
7. Garcia, E., Casbeer, D.W., von Moll, A., Pachter, M.: Multiple pursuer multiple evader differential games. *IEEE Transactions on Automatic Control* **66**(5), 2345–2350 (2020)
8. Guan, Y., Maity, D., Kroninger, C.M., Tsiotras, P.: Bounded-rational pursuit-evasion games. In: *American Control Conference (ACC)*. pp. 3216–3221. New Orleans, LA (2021)
9. Guan, Y., Zhang, Q., Tsiotras, P.: Learning Nash equilibria in zero-sum stochastic games via entropy-regularized policy approximation. *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, Montreal, QC, Canada (2021)
10. Hart, S., Mas-Colell, A.: A simple adaptive procedure leading to correlated equilibrium. *Econometrica* **68**(5), 1127–1150 (2000), <http://www.jstor.org/stable/2999445>

11. Jain, M., Korzhyk, D., Vaněk, O., Conitzer, V., Pěchouček, M., Tambe, M.: A double oracle algorithm for zero-sum security games on graphs. In: The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 1. Taipei, Taiwan. p. 327–334. AAMAS '11, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2011)
12. Kuhn, H.W.: The Hungarian method for the assignment problem. *Naval research logistics quarterly* **2**(1-2), 83–97 (1955)
13. Li, S., Wang, X., Zhang, Y., Xue, W., Černý, J., An, B.: Solving large-scale pursuit-evasion games using pre-trained strategies. In: Proceedings of the AAAI Conference on Artificial Intelligence. Washington, DC, USA,. vol. 37, pp. 11586–11594 (2023)
14. Nijssen, J.A.M.: Monte-Carlo tree search for multi-player games. Ph.D. thesis, Maastricht University, Maastricht, The Netherlands (2013). <https://doi.org/0.26481/dis.20131202jn>
15. Parsons, T.D.: Pursuit-evasion in a graph. In: Theory and Applications of Graphs: Proceedings, pp. 426–441. Springer (2006)
16. Schmid, M., Moravčík, M., Burch, N., Kadlec, R., Davidson, J., Waugh, K., Bard, N., Timbers, F., Lanctot, M., Holland, G.Z., et al.: Student of Games: A unified learning algorithm for both perfect and imperfect information games. *Science Advances* **9**(46), eadg3256 (2023)
17. Shishika, D., Kumar, V.: Local-game decomposition for multiplayer perimeter-defense problem. In: IEEE Conference on Decision and Control (CDC). Miami Beach, FL, USA. pp. 2093–2100 (2018). <https://doi.org/10.1109/CDC.2018.8618879>
18. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., Hassabis, D.: A general reinforcement learning algorithm that masters Chess, Shogi, and Go through self-play. *Science* **362**(6419), 1140–1144 (2018). <https://doi.org/10.1126/science.aar6404>
19. Świechowski, M., Godlewski, K., Sawicki, B., Mańdziuk, J.: Monte Carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review* **56**(3), 2497–2562 (2023)
20. Tak, M.J.W., Lanctot, M., Winands, M.H.M.: Monte Carlo tree search variants for simultaneous move games. In: IEEE Conference on Computational Intelligence and Games. Dortmund, Germany,. pp. 1–8 (2014). <https://doi.org/10.1109/CIG.2014.6932889>
21. Theodoridis, A., Chalkiadakis, G.: Monte Carlo tree search for the game of Diplomacy. In: 11th Hellenic Conference on Artificial Intelligence. Athens, Greece,. pp. 16–25 (2020)
22. Xue, W., An, B., Yeo, C.K.: NSGZero: Efficiently learning non-exploitable policy in large-scale network security games with neural Monte Carlo tree search. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 36, pp. 4646–4653 (2022)
23. Yu, C., Velu, A., Vinitsky, E., Gao, J., Wang, Y., Bayen, A., Wu, Y.: The surprising effectiveness of PPO in cooperative, multi-agent games (2022), <https://arxiv.org/abs/2103.01955>
24. Zhuang, S., Li, S., Yang, T., Li, M., Shi, X., An, B., Zhang, Y.: Solving urban network security games: Learning platform, benchmark, and challenge for AI research (2025), <https://arxiv.org/abs/2501.17559>