Elastic Monte Carlo Tree Search

Linjie Xu, Alexander Dockhorn, Diego Perez-Liebana

***Abstract*—Strategy games are a challenge for the design of AI agents due to their complexity and the combinatorial search space they produce. State abstraction has been applied in different domains to shrink the search space. Automatic state abstraction methods have gained much success in the planning domain and their transfer to strategy games raises a question of scalability. In this paper, we propose Elastic MCTS, an algorithm that uses automatic state abstraction to play strategy games. In Elastic MCTS, tree nodes are clustered dynamically. First, nodes are grouped by state abstraction for efficient exploration, to later be separated for refining exploitable action sequences. Such an elastic tree benefits from efficient information sharing while avoiding using an imperfect state abstraction during the whole search process. We provide empirical analyses of the proposed method in three strategy games of different complexity. Our empirical results show that in all games, Elastic MCTS outperforms MCTS baselines by a large margin, with a considerable search tree size reduction at the expense of small computation time. The code for reproducing the reported results can be found at https://github.com/GAIGResearch/Stratega.**

***Index Terms*—State Abstraction, MCTS, Strategy Games.**

I. Introduction

Strategy games are challenging for AI players due to their large and complex search spaces. Human expert players can quickly learn effective strategies to defeat built-in AI in this type of games. These agents typically use different methods for decision-making, such as heuristics or scripts created by game designers with incorporated domain knowledge. Search-based AI agents guided by heuristics [1] can identify good trajectories with a short horizon (or search depth), while agents combined with scripts scale better in complex games [2]. However, those scripts are usually rigid and limited by the creator's ability to capture the complexity of decision-making (as in most rule-based systems). These *static* scripts cannot adapt to game modifications, typical during the game development cycle. In this case, these scripts need repeated updates, requiring extra workload for designers and developers.

While heuristics and scripts are created using expert knowledge, agents can also derive game knowledge using a forward model. A forward model is a simulator that allows agents to sample different future playing trajectories, to then model this knowledge in a data structure such as a tree. A forward model returns a possible next state given a state-action pair. By simulating future time steps, the agent finds a strategy that guides the current move. Monte Carlo Tree Search [3] (MCTS) is a popular planning method that balances exploration and exploitation while building a search tree. However, strategy games with large search spaces, often increasing combinatorially with the number of units under control, make search-based methods struggle to find good strategies in a reduced time frame. Here, and in most domains, improving sampling efficiency is crucial for good performance.

Abstraction [4] is technique for reducing the search space. In strategy games such as StarCraft, existing works on state abstraction [5], [6] and action abstraction [2], [7], [8] have shown to outperform non-abstracting methods. The abstractions used in these works depend on domain knowledge for specific tasks, hence they fail to be applied to new games. In the planning domain, more general abstractions that require no domain knowledge are extensively studied [9]–[15], often in tasks of relatively low complexity. In our work, we aim to evaluate whether similar concepts from the planning domain can be scaled up to strategy games.

In this paper, we propose an algorithm that employs state abstraction on MCTS to play strategy games. The state abstraction aggregates tree nodes by approximate homomorphism [16] for Markov Decision Process (MDP), to reduce the size of the tree. The challenges derived from implementing this approach are discussed in this paper and solutions are proposed to address them. In our previous work [17], we proposed two solutions. First, a modification of MCTS (which we call MCTS$_u$) is implemented to simplify the action space. In MCTS$_u$, each node is assigned to a unit, and therefore it considers actions available for one unit rather than for all of them. This reduces the number of samples required for an abstraction. Secondly, an interaction threshold $\alpha_{ABS} \in \mathbb{N}$ is introduced to indicate a stopping iteration for the use of abstraction. As the MCTS iterations $N_{mcts} \in \mathbb{N}$ reaches α_{ABS}, the state abstraction that aggregated tree nodes in previous iterations is abandoned and the tree is "expanded" (i.e. abstract nodes are eliminated), continuing its search as in standard MCTS. Given that the tree expands and shrinks during the search, we call our algorithm Elastic MCTS.

This paper extends the previous work [17] by including three baselines and two more games. Two of these baselines are Linear Side Information (LSI) [18] and Naive MCTS [19], which are are canonical methods designed for sequential decision-making problems with combinatorial action spaces. The third baseline is RG MCTS$_u$ which, instead of using MDP homomorphism to group nodes in the tree, employs a random selection of nodes to cluster them. The objective of this addition is to test if the benefits observed in Elastic MCTS are due to using *any* grouping of nodes, or the one suggested by MDP homomorphism. Additionally, we add two games, *Push them All* and *Two Kingdoms*, to test Elastic MCTS under different scenarios, dynamics and action space complexities, in order to evaluate the generality of our approach.

Our contributions are summarized as follows:

- **Automatic state abstraction for strategy games with no domain knowledge:** Our method applies a state abstraction that requires no domain knowledge for complex environments such as strategy games, in contrast to existing methods which require domain knowledge.

While this work focuses on strategy games, the method proposed in this paper may be applicable to other genres, as it does not require game-specific knowledge.

- **An analysis on the effects of state abstraction on the recommendation policy:** Previous works keep the generated abstraction within the tree of MCTS during all iterations. This approach is based on the assumption that the policy resulting from the abstraction is better than the policy from the original state space, neglecting the risk of using a bad-quality state abstraction. Our algorithm sets up an iteration threshold for using the abstractions, which we tune to analyze the impact of turning back to the original tree at different times during the search.
- **Generality and validation of Elastic MCTS:** The algorithm proposed here outperforms all the other compared methods in two of the three games tested and outperforms all other methods based on MCTS with unit ordering in all three games. Importantly, it outperforms the version of Elastic MCTS that groups nodes at random in all cases, showing that the reason behind the performance boost observed is due to the use of MDP homomorphism rather than *any* reduction of the search tree size, as achieved by the random group selection performed by RG MCTS$_u$.

II. Background

A. Monte Carlo Tree Search (MCTS)

MCTS [20] [3] generates a search tree to estimate state-action values. The root node represents the given state, whereas nodes and branches represent states and actions, respectively. In tree nodes, the cumulative reward X and the visit count N are stored and updated while building the tree. MCTS builds the tree iteratively, each iteration including 4 steps: selection, expansion, simulation, and back-propagation.

The *selection* phase returns a leaf node by traversing the tree from the root until a not-fully expanded node is reached. During this phase, a *tree policy* is used to select the next node. A canonical choice for the tree policy is Upper Confidence Bounds (UCB), among which the UCB1 is widely used. The UCB1 value for each node is:

$$UCB1(s,a) = \frac{X(s,a)}{N(s,a)} + C\sqrt{\frac{\ln N_{\text{parent}}}{N(s,a)}}, \tag{1}$$

where s is the state represented by the corresponding node, a the action taken at s, $X(s,a)$ the cumulative reward, $N(s,a)$ the node's visit count, and N_{parent} the parent node's visit count. In UCB1, a constant $C \in \mathbb{R}$ controls the exploration-exploitation trade-off. The tree node with the highest UCB1 value will be selected to descend the tree until a non-fully expanded node is reached, when a new child node is generated in the *expansion* phase. The *simulation* phase starts from the previously expanded node, using a roll-out policy to sample actions until a fixed depth is reached or until the game ends. Often, this roll-out policy consists of a uniform random selection of available actions. The final state visited in the simulation is then evaluated, and its value (the game's outcome) is *back-propagated* through the nodes visited in this iteration until the root node. For non-terminal states [21], [22], a heuristic function is used instead. When the budget runs out, a *recommendation policy* is used to pick an action for execution (e.g. the most visited child of the root node).

B. State Abstraction and Approximate MDP Homomorphism

A Markov Decision Process (MDP) is defined as $\langle \mathbb{S}, \mathbb{A}, R, T, \gamma \rangle$, where $\mathbb{S}$ is the state space, $\mathbb{A}$ the action space, $R : \mathbb{S} \times \mathbb{A} \mapsto \mathbb{R}$ the reward function, $T : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \mapsto [0,1]$ the transition probability function and $\gamma \in \mathbb{R}, 0 < \gamma < 1$ is a discount factor. State abstraction of an MDP can be formalized with a mapping $\phi : \mathbb{S} \mapsto \mathbb{S}_\phi$ from state space to an abstract state space $\mathbb{S}_\phi$, preferably of smaller size. Each element of $\mathbb{S}_\phi$ is an abstract state consisting of one or many ground state(s). Approximate MDP homomorphism [16] defines a similarity measure between states (s_1, s_2) by two approximation errors:

$$\epsilon_R(s_1,s_2) = \max_{a \in \mathbb{A}} |R(s_1,a) - R(s_2,a)| \tag{2}$$

$$\epsilon_T(s_1,s_2) = \max_{a \in \mathbb{A}} \sum_{s'_\phi \in \mathbb{S}_\phi} | \sum_{s' \in s'_\phi} T(s'|s_1,a) - \sum_{s' \in s'_\phi} T(s'|s_2,a)|. \tag{3}$$

The reward function error $\epsilon_R(s_1,s_2)$ measures the maximum $l1$ distance of reward between two states. The transition probability error $\epsilon_T(s_1,s_2)$ measures the worst-case total variation distance between two states conditioned by action and falls in the range $[0,2]$. In MCTS, nodes from the same depth are aggregated together if the two corresponding states have reward function error and transition probability error both below the threshold, i.e., $\epsilon_R(s_1,s_2) \leq \eta_R$ and $\epsilon_T(s_1,s_2) \leq \eta_T$. The node group is called *abstract* node and the abstraction at each depth is a local approximate MDP homomorphism. We note the reward and visit count of an abstract node is shared between all its ground nodes (tree nodes generated by traditional MCTS), denoted by $\hat{X} = \frac{\sum X_i}{m}$ and $\hat{N} = \frac{\sum N_i}{m}$, where m is the size of this abstract node, i.e., the number of nodes aggregated. These statistics are updated when a new ground node is added to an abstract node.

III. Related Work

In strategy games, state abstraction has been widely used to obtain more efficient agents. [23] proposed to group MCTS tree nodes that have similar game-specific features for efficient searching in computer GO. [24] proposed to use a hand-crafted state abstraction to reduce map complexity by dividing the game map into tiles. This state abstraction enhanced the Monte Carlo algorithm in playing the *Capture The Flag* game. In [25], state abstraction was applied to encode the Starcraft map into regions connected by checkpoints, largely simplifying the state space. This state abstraction was combined with action space pruning in [5], to play combat scenarios. In their high-level game state representation, combat-irrelevant units such as workers and buildings were eliminated to reduce the search space, resulting in an agent showing a performance close to script-based agents. [26] decomposed the game state to unit vectors that contain unit information, eliminating superfluous information and grouping similar nodes. While these state

abstractions enhanced the agent's game-playing performance, most of them were generated with domain knowledge, limiting their application to new games.

In the planning domain where the problem is less complex than in strategy games, state abstractions that require no domain knowledge were widely studied. Eyck et al. [27] empirically studied node grouping both with or without prior knowledge. Their work has shown that random grouping, grouping according to payoffs, and grouping with prior knowledge have different influences on the searching speed and final performance, even though all these grouping methods reduce the branching factor. Progressive Abstraction Refinement for Sparse Sampling (PARSS) [12], [14] proposed to start with a coarse abstraction, in which all the states are clustered in the same group. This group is gradually divided into different groups to refine the state abstraction. *On-The-Go* abstraction [13] updates the abstraction more frequently, avoiding the influence of delayed samples. A *recent visit count* is kept for each node and the abstraction mapping for a node is re-computed once the *recent visit count* reached a threshold.

Jiang et al. [9] proposed to construct state abstraction for MCTS with approximate MDP homomorphism. In each depth of MCTS, tree nodes that have a similar transition probability function and reward function are grouped. Their similarities are defined as in Eqs. 2 and 3. After every batch of MCTS iterations, a state abstraction is generated from collected samples. The proposed method improved the agent's performance in the game Othello when combined with further action abstraction and tree pruning. [11] extended the application of approximate MDP homomorphism from state abstraction to state-action abstraction where the state-action pairs are grouped. In our paper, we adapt approximate MDP homomorphism from [9] to perform in the more challenging domain of strategy games, without providing game expert knowledge.

There are methods designed to find solutions for strategy games where the action space is a composite of unit action spaces. Previous approaches that adopt Combinatorial Multi-armed Bandit (CMAB) achieved success in this problem. Naive MCTS [19] integrated CMAB into the selection phase to recommend action combinations. Linear Side Information (LSI) [18] proposed a two-phase algorithm where a set of candidate action combinations are generated by evaluating each action and then the final action combination is recommended using an MAB. Different from these approaches that search for an action combination, our method addresses combinatorial action space by reducing selection space for each unit.

IV. Stratega

To evaluate agents in different strategy games, we use the Stratega [28] framework where new games can be quickly implemented. The battlefield in Stratega is depicted in an isometric view, where each tile can hold one or several elements such as landforms, buildings, resources, and army units (see Figure 1). One of the key features of Stratega is that a forward model is provided to support statistical forward planning methods, such as MCTS or Rolling Horizon Evolution [29]. We use three games from Stratega – *Kill The King (KTK)*, *Push Them All (PTA)*, and *Two Kingdoms (TK)* – as evaluation for our proposed method.

Fig. 1: An example of a game in the Stratega framework.

In *KTK*, there are 4 unit types: *King, Warrior, Archer* and *Healer*. *King, Warrior* and *Archer* have similar action types: [*Move, Attack, Do-nothing*]. The action types for *Healer* are: [*Move, Heal, Do-nothing*]. Action types are parameterized, which affects action space sizes. For example, the *Move Range* parameter for action type *Move* is set to 2 for the *King* unit, resulting in 12 surrounding tiles as possible targets. Its *Attack Range* for Attack is set to 2, allowing any opponent unit within 2 tiles to be targeted. In this setting, the maximum size of its action space is $(12 + 1) \times (12 + 1) = 169$, as units are allowed to first move ($+1$ for not moving) and then attack ($+1$ for not attacking). The whole action space for a player with many units is a combinatorial space bounded at $(\max|\mathbb{A}_u|)^n$, where the $\mathbb{A}_u$ is the size of action space of any unit on the board. For *KTK*, we experiment with various numbers of units between 4 and 11, which constitutes an action space bounded between 10^5 and 10^{14} actions. The combination of different unit compositions and their diverse parameterization and their resulting action sets yields many strategies. The attribute set for this game consists of *Health Points, Move Range, Attack Range, Attack Damage, Heal Strength*. Additionally, areas of mountains and water are used to create various layouts, which require units to coordinate their actions to be effective.

In *PTA*, there is only 1 unit type: *Pusher*, who can push one adjacent unit in different directions. To win this game, the player controls units to push all opponent units into holes distributed across the map. In our configuration, each player has 3 pushers. With the size of action space for each unit as $(4 + 1) \times (4) \times 4 = 80$ (move to 4 neighbor positions or not moving, push a neighbor unit, and push the target unit in 4 different directions). The size of the resulting action space is $80^3 = 512,000$. *PTA* differs from *KTK* in its dynamics (units can change their positions as well as the opponents') and challenges (e.g. positioning coordination between units).

TK is designed for evaluating whether the Elastic MCTS can scale to more complex strategy games. *TK* is a combat game with researching, unit spawning, resource collection, and resource management. This game shares the same goal as *KTK*, killing the opponent King. However, units are required to be spawned and their cost and conditions vary. There are two types of cost, including gold (to be collected from gold veins), and production, which increases in every turn. Specific technologies must be researched to spawn units. e.g. *Mining*

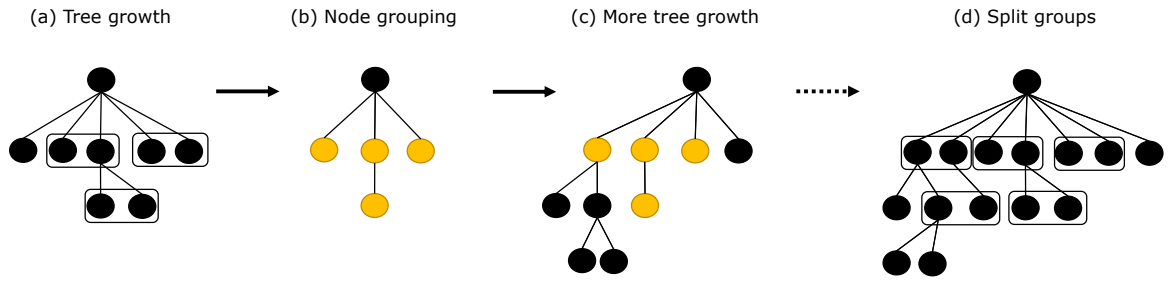


Fig. 2: Overview of dynamic changes of tree nodes in Elastic MCTS. Ground nodes are *black* while abstract nodes are *yellow*. After a number of iterations (a), ground nodes are grouped by using Approximate MDP Homomorphism (b). Search continues adding more ground nodes (c), repeating state aggregation after every B iterations. When the abstraction iteration threshold is reached, abstract nodes are split (d) and the search continues to explore the original game without using the state abstraction.

must be researched to spawn a worker. In this game, the unit types are defined as: *King, Castle, Worker, Warrior, Knight, Wizard* and *Healer*. At the beginning of the game, each player has only a castle and a king. The player spawns workers to collect gold from gold veins and use it to spawn armies to protect its King and attack the opponent.

The action space for *TK* varies largely since units can be spawned. There is a minimum action space size (at the beginning of the game) that is $169 + 1$, including the action space for the $King$ and whether to spawn a worker. The maximum action space size depends on how many units are there on the map. See for example a game where one worker and two warriors were spawned. The action space for a king or a warrior is a composition of move and attack: $(12+1) \times (12+1) = 169$. For a worker, the action space is a composition of movement and mining: $(12+1) \times (12+1) = 169$. There are also spawning unit actions (at the 8 tiles surrounding the *Castle*) for each unit: $5 \times 8 = 40$. There is a choice of whether to research Mining, resulting in 2 actions. The resulting action space is $169^4 \times 40 \times 2 \approx 6 \times 10^{10}$.

All these games have a maximum round limit. When this limit is reached and no player has achieved the condition to win, the game results in a draw. The maximum round limits for *KTK, TK* and *PTA* are $100, 100$, and 200, respectively. We choose these three games to evaluate our proposed algorithm for their different characteristics. *KTK* and *TK* share many challenges with Starcraft even though their battles are not as complex. Firstly, *KTK*, *TK*, and Starcraft all have combinatorial state and action spaces, which is a common feature in most strategy games. *KTK* focuses on how to optimize the strategy for a given unit set and TK requires the agent to control units for different goals (e.g. collecting resources or attack) and decide which units to be included in its army. *PTA* aims at evaluating state abstractions in more tactical scenarios.

V. Elastic MCTS

We propose *Elastic MCTS* for strategy games, which constructs state abstractions for MCTS with approximate MDP homomorphism. First, we introduce two modifications that are needed to adapt the principles of MDP homomorphism to the combinatorial search spaces present in strategy games: MCTS with unit ordering (units move sequentially) to reduce the action space (Section V-A) and introducing an iteration threshold to revert to the original state space (V-B). We assume a multi-unit setting, but our method can be used also for single-unit games. In the single-unit setting, our method differs from [9] by the node un-grouping step. Our final method is described in detail in Section V-C and depicted in Fig. 2.

A. Approximate Homomorphism in Strategy Games

Our method is based on the state abstraction proposed by Nan Jiang [9], where a local approximate MDP homomorphism (see Section II-B for details) is constructed and constantly updated from trajectories sampled by MCTS. As this method is originally designed for the planning domain, two issues occur when it is applied to complex search spaces such as those from strategy games.

First, the size of samples required to generate a good-quality approximate homomorphism depends on the sizes of the state and action spaces. This requirement becomes infeasible in strategy games, where the state space grows exponentially and the action space is highly variable. Secondly, the calculation of reward and transition approximation errors ϵ_R and ϵ_T (Eqs. 2 and 3) requires the execution of all possible actions available in two states. For two states that have different action spaces (common in strategy games), the original definition of ϵ_R and ϵ_T can not apply. While using only the small set of common actions would possibly resolve the approximation errors, the resulting approximate errors would not represent the true similarities of both states in terms of rewards and transitions. Moreover, this solution will lose the performance guarantees provided by approximate MDP homomorphism [9].

We alleviate these two issues by developing a variant of MCTS called MCTS$_u$ (MCTS with unit ordering). In MCTS$_u$, each node corresponds to a game state in which a single unit is being controlled. This is similar to the hierarchical expansion of action in [30]. Thus, the action space for each node is much smaller than the original combinatorial one. Additionally, nodes in MCTS$_u$ have a large proportion of common available actions in states where the same unit acts. To implement MCTS$_u$, we can fix the units' move order. Inside MCTS$_u$, the move order is set randomly at the beginning of the game and kept fixed. When a new unit is spawned, it moves after all other units move. While, theoretically, the

guarantee of MCTS that converges to minimax does not hold anymore, agents under this setting are empirically found as strong players with advantages obtained by the reduced action space. The results and discussion on this are in Section VI-B and are in line with previous results from the literature [31].

B. Elastic State Grouping and Un-grouping

We construct a state abstraction from samples collected by MCTS to group tree nodes. The state abstraction is generated in a batch manner: for every consecutive B iterations of MCTS, an approximate homomorphism is constructed from the sampled trajectories to aggregate similar nodes into groups, according to the states represented by these nodes. In MCTS, each tree node stores statistics including cumulative reward and visit count. For a node group, its statistics are drawn by averaging statistics from its ground nodes: the UCT value of one node belonging to a node group is calculated based on its group statistics, and the group node statistics are updated during the back-propagation step.

Existing methods [9], [11], [13] assume the generated state abstraction to be of high quality so they keep using the state abstraction until the search is finished. However, it is well known that approximate MDP homomorphism is a lossy abstraction, i.e., introduces imperfections in the search space [13]. With a lossy abstraction, the optimal policy obtained from abstraction performs worse than the optimal policy derived from the original search space. For strategy games, MCTS$_u$ reduces the size of action space and helps construct a better abstraction, but these imperfections remain. Additionally, states belonging to the same group have common statistics, forcing the action selection for the recommendation policy to choose effectively at random among actions that lead to a child node (but a different ground node).

We propose a novel approach of using state abstraction to tackle both problems mentioned above. We set up an iteration threshold α_{ABS} for using state abstraction, i.e, the MCTS runs with state abstraction for α_{ABS} iterations. After that, MCTS assigns the statistics of node groups to their member ground nodes, and the state abstraction is abandoned (breaking the node groups into ground tree nodes). For the remaining budget, the algorithm runs as traditional MCTS. After α_{ABS} iterations, MCTS does not search in an imperfect space, and nodes that belonged to the same group might now be independently visited. This allows MCTS's recommendation policy to distinguish better among the available actions.

C. Elastic MCTS

We present Elastic MCTS in three separate parts, with pseudocode in Algorithms 1, 2 and 3. Algorithm 1 shows the general framework of Elastic MCTS that runs MCTS iterations (line 3) and controls the state abstraction until the forward model calls budget (N_{fm}) is exhausted (line 2). Each rollout step uses one unit of this budget, where a state and an action are provided to the forward model to retrieve the next state. The abstraction is initialized to map states to themselves (line 1). When the number of iterations surpasses the threshold α_{ABS} (line 4), the state abstraction is abandoned to return to the original group search space (line 5), by splitting all node groups to ground nodes and assigning group statistics to each member node. This procedure is also depicted in Figure 2.

Algorithm 1 Elastic MCTS. $N_{fm} \in \mathbb{N}$: forward model calls budget. $\alpha_{ABS} \in \mathbb{N}$: MCTS iteration threshold. $N_{mcts} \in \mathbb{N}$: current MCTS iteration. $B \in \mathbb{N}$: batch size. $\phi(s) = \hat{s}$: state abstraction mapping a ground node s to a node group $\hat{s}$. $\eta_R, \eta_T \in \mathbb{R}$ are reward and transition error thresholds, resp.

Require: $N_{fm}, \alpha_{ABS}, \eta_R, \eta_T$
1: $\phi := s \to \hat{s}, \hat{s} = \{s\}$ # Initialize the abstraction
2: **while** $n_{fm} < N_{fm}$ **do**
3: $MCTSIteration(\phi)$
4: **if** $N_{mcts} > \alpha_{ABS}$ **then**
5: $\phi := s \to \hat{s}, \hat{s} = \{s\}$ # Fig.2 (d)
6: **else if** $N_{mcts} \% B == 0$ **then**
7: $\phi = ConstructAbstraction(\phi, \eta_R, \eta_T)$ # Fig.2 (b)
8: $N_{mcts} = N_{mcts} + 1$

Algorithm 2 MCTSIteration(ϕ)

1: **while** true **do**
2: Select a child K that maximizes $V_{uct} = X(\phi(s), a)/N(\phi(s), a) + C\sqrt{\ln N_{parent}/N(\phi(s), a)}$.
3: **if** Node K is not fully-expanded **then**
4: Expand this node by generating a new child node P.
5: Rollout for P and obtain reward R.
6: **break**
7: **else if** Node K represents the end of the game **then**
8: Obtain reward R from sate evaluation function.
9: **break**
10: Backpropagate R, updating $X(s, a)$ and $N(s, a)$ for node group $\phi(s)$ in selection path.

Algorithm 2 shows the 4 phases of MCTS, but differing from normal MCTS in that it uses cumulative reward and visit count from the node group rather than the ground tree nodes, in the selection (line 2) and back-propagation (line 10) steps. Nodes within the same node group have similar probabilities to be selected as they share statistics. New nodes added to the tree in the expansion phase are added as ground nodes, merging into groups after B iterations, as shown in Algorithm 1.

Algorithm 3 shows the update of the state abstraction: from the deepest layer to the root (line 1), for every ground tree node that is not a member node of a node group (line 2), a similarity check is performed against all sibling node groups. This similarity is measured by computing two approximate errors ϵ_R and ϵ_T of approximate MDP homomorphism. The samples collected by MCTS are triplets and each of them contains a state, an action, and a return: $< s, a, R >$. To compute the ϵ_R error between two states s_1 and s_2, $|R(s_1, a) - R(s_2, a)|$ is calculated for each action a that is available for both s_1 and s_2, with $R(s_i, a) = 0$ if a is invalid for any s_i. ϵ_R takes the value of the maximum difference found among actions (line 7). The computation of ϵ_T also requires iterating all actions. The approximate error $|T(s'|s_1, a) - T(s'|s_2, a)|$ equals 1 only when both s_1 and s_2 have action a as their valid action, and this action leads to the same next state s'. In this work, we

Algorithm 3 ConstructAbstraction(ϕ, η_R, η_T), l is the tree depth and L is the maximum depth of the current tree.

1: **for** $l = L$ to 1 **do**
2: **for all** state s_1 in depth l that is not grouped **do**
3: s_1_grouped = false
4: **for all** abstract state $\hat{s}$ in ϕ_l of depth l **do**
5: s_1_in_$\hat{s}$ = true
6: **for all** state s_2 in $\hat{s}$ **do**
7: $\epsilon_R = \max_a |R(s_1, a) - R(s_2, a)|$
8: $\epsilon_T = \sum_{s'} |T(s'|s_1, a) - T(s'|s_2, a)|$
9: **if** $\epsilon_R > \eta_R$ **or** $\epsilon_T > \eta_T$ **then**
10: s_1_in_$\hat{s}$ = false, **break**
11: **if** s_1_in_$\hat{s}$ == true **then**
12: Add s_1 in abstraction node
13: s_1_grouped = true, **break**
14: **if** s_1_grouped = false **then**
15: Create a new abstract node

only consider clustering nodes that represent the same unit, their next state s'_1 and s'_2 are recognized as identical when the unit-related attributes have the same values. If an existing node group is found where $\epsilon_R \leq \eta_R$ and $\epsilon_T \leq \eta_T$ for a candidate ground node, we add the ground node to its member nodes (lines 11-13). If this ground node is found no group fulfills this condition, a new group is created where the only member node is this ground node (line 15).

VI. Experimental Work

Our experimental setup spans through three games (described in Section IV) and five agents[1]:

1) *Rule-based Agents*: based on built-in rule-based agents from Stratega [28]. The strategy for *KTK* rule-based agent is to concentrate attacks on a single isolated enemy unit and assign healers to heal the strongest ally units. For *PTA*, each pusher moves to the nearest enemy and pushes the enemy towards the nearest hole. In *TK*, the strategy is spawning one worker that moves to the nearest gold vein for collecting gold, to then spawn *Warriors* that follow the same strategy as in *KTK*.
2) *MCTS*: the default MCTS algorithm.
3) *MCTS$_u$*: MCTS with unit ordering, no state abstraction.
4) Naive MCTS: an MCTS algorithm that applies Naive Sampling [19] to the selection and expansion stages.
5) LSI: LSI algorithm [18] evaluates unit actions to generate a set of action combinations, then recommend the final action combination by solving an MAB problem.
6) *Elastic MCTS$_u$*: Elastic MCTS with fixed unit order.
7) *RG MCTS$_u$* : Elastic MCTS with fixed unit order, where same-depth nodes from the same depth are grouped uniformly at random (joining one of the available groups or creating a new one), rather than using MDP homomorphism.

For each game, we employ different *state evaluation functions*, used by non-rule-based algorithms. All these functions give a reward of $0.0 \leq R \leq 1.0$ to a state after normalization. The rewards for win, loss, and draw are $1, -1$, and 0, respectively. If the given state is not terminal, the value of the state is assigned with a task-specific heuristic. In *KTK*, the value of the state is $R = 1 - \frac{d \cdot h}{D \cdot H}$, where d is the average distance between all player's units and the opponent's king (D: maximum possible distance) and h the health point of opponent's king (H: the maximum health). This function encourages getting close to the opponent's king and attack it.

In *PTA*, The rewards for win, loss are 1, and 0, respectively. For a non terminal state or a draw state, a score in $[0, 1]$ is computed in three parts: Firstly, $0.2 \times \frac{\sum_u \min_{u'} dis(u,u')}{D}$, where u is a player's unit, u' is an opponent's unit, $dis(\cdot, \cdot)$ is the euclidean distance, and D is the largest distance in a map. Secondly, $0.4 \times \frac{|U_t|}{|U_0|}$, where $|U_t|$ is the number of player's units and $|U_0|$ is the number of player's units at the start. Finally, $0.4 \times \frac{|U'_0| - |U'_t|}{|U'_0|}$, where $|U'_t|, |U'_0|$ are the numbers of opponent units at the current step and at the start, respectively.

For *TK*, the rewards for win, loss are 1, and 0, respectively. For non-terminal states including draw states, scores are calculated as a weighted sum of different state attributes: $0.2\times$ whether *Mining* is researched, $0.1\times$ whether the player has workers, $0.1\times$ whether the player has a unit that can attack, $0.1 \times (\max_{d_w} - d_w)$, with d_w being the distance between workers to the nearest gold vein, $0.2\times$ the gold possessed by the player, and $0.3 \times (\max_d - d)$, with d being the distance between the player's units and the opponent's units. All values are normalized to ensure the final scores land in $[0, 1]$.

A. Parameter Optimization for Agents with NTBEA

All our experiments evaluate agents that are pitched against each other in $1vs1$ games. For fair comparison, we use the N-Tuple Bandit Evolutionary Algorithm (NTBEA) [32] to tune the parameters of each MCTS agent independently. NTBEA utilizes a combinatorial multi-armed bandit to navigate the parameter space while building a landscape model for noisy evaluation functions. NTBEA also has its own parameters: we set the exploration factor for the multi-armed bandit to 2, we use 50 neighbors, and 50 iterations. When tuning parameters for all agents, we used the win rate of the agent playing against *Rule-based Agent* as fitness. For the agents *MCTS*, MCTS$_u$, Elastic MCTS$_u$, and RG MCTS$_u$, the budget is set as the number of forward model calls. For *KTK, PTA* and *TK*, their corresponding budget is set as $10,000; 10,000$, and $5,000$ respectively, with which the MCTS is able to perform competitively against the different rule-based agents.

MCTS and MCTS$_u$ have the same parameter spaces for NTBEA: we evaluate the exploration factor $C \in \{0.1, 1, 10, 100\}$ and rollout length $L \in \{10, 20, 40\}$. Elastic MCTS$_u$ adds to these two parameters (C and L) the reward function error $\eta_R \in \{0.0, 0.05, 0.1, 0.3, 0.5, 1.0\}$, the transition probability error $\eta_T \in \{0.0, 0.5, 1.0, 1.5, 2.0\}$ and the iteration threshold to stop using abstractions $\alpha_{ABS} \in \{4\times B, 8\times B, 10\times B, 12\times B\}$, where B is the batch size set to 20 for all experiments. RG MCTS$_u$ adds to C and L the $\alpha_{ABS} \in \{4 \times B, 8 \times B, 10 \times B, 12 \times B\}$ controlling when to split node groups. The resulting tuned parameters for all agents and games are listed in Table I.

[1]Framework and all agents available at github.com/GAIGResearch/Stratega

TABLE I: Hyper-parameters for agents tuned by NTBEA

Game	Agents	C	L	α_{ABS}	η_R	η_T
KTK	MCTS	0.1	10	/	/	/
	MCTS$_u$	1.0	10	/	/	/
	Elastic MCTS$_u$	0.1	10	$10B$	0.05	1.0
	RG MCTS$_u$	0.1	10	$8B$	/	/
PTA	MCTS	10	10	/	/	/
	MCTS$_u$	10	20	/	/	/
	Elastic MCTS$_u$	10	10	$8B$	1.0	1.0
	RG MCTS$_u$	0.1	10	$4B$	/	/
TK	MCTS	0.1	20	/	/	/
	MCTS$_u$	1.0	20	/	/	/
	Elastic MCTS$_u$	1.0	20	$6B$	0.05	1.0
	RG MCTS$_u$	0.1	10	$8B$	/	/

The LSI agent has two parameters to tune: N_g and N_e, which are the budgets for the generation and evaluation stages in LSI, respectively. To set the same budget as the other methods, 6 combinations of $\{N_g, N_e\}$ with the same cost are generated per game. Linear search is applied to recommend one parameter combination, resulting in the usage of the following LSI parameters (N_g, N_e) in our experiments: $(40, 150)$, $(120, 80)$ and $(30, 80)$ for *KTK, PTA* and *TK*, respectively.

For Naive MCTS, the tunable parameters space are defined as $\epsilon_g = \{0.7, 0.9\}, \epsilon_l = \{0.3, 0.5, 0.7\}$ and rollout length $L \in \{10, 20, 40\}$. We use NTBEA to optimize one parameter combination for each game, resulting in the usage of the following parameters $(\epsilon_g, \epsilon_l, L)$ in our experiments: $(0.9, 0.7, 20)$ for *KTK*, $(0.9, 0.7, 10)$ for *PTA* and $(0.7, 0.7, 20)$ for *TK*.

B. Algorithmic Performance

This section describes the results of our experimental work. We report win rates for each player in a match (draw rates can be inferred). Each win rate is shown with its standard error among 5 game runs with 5 different random seeds. With each seed, the game is run 100 games (50×2 due to side switching). This resulted in total 500 games for each pair of agents. Units of both players are placed symmetrically in all maps.

Performance in Kill The King: we first evaluate the proposed method, by testing agents playing against each other in different variants of *KTK*. We designed two groups of experiments. The first group evaluates the presented agents across scenarios with different amounts of units to observe the effect of different action space sizes. The second group investigates the performances in different game maps.

For the first group evaluations, we generate 50 instances with the same asymmetric map ("lak110d" from https://movingai.com/ [33]) with different random initial unit positions. As the goal for this evaluation is to explore unit numbers, we generate 3 army compositions: *(1 King, 1 Warrior, 1 Archer, 1 Healer), (1 King, 2 Warriors, 2 Archers, 2 Healers) and (1 King, 3 Warriors, 3 Archers, 3 Healers)*. Tables II, III and IV show the experimental results for these 3 settings. As shown, in all army compositions Naive MCTS, MCTS$_u$, Elastic MCTS$_u$ and RG MCTS$_u$ clearly outperform the *Rule-based Agent* while the performance of MCTS varies among different army compositions. LSI fails to beat the *Rule-based Agent* in all settings of the game. When playing against MCTS, Naive MCTS, MCTS$_u$, Elastic MCTS$_u$ and RG MCTS$_u$ all outperform MCTS by large margins. Among these agents, Elastic MCTS$_u$ shows the highest win rates. These results confirm that unit ordering helps gain much performance boost even though these unit orders are randomly picked.

TABLE II: Win rates with standard errors for the game *Kill The King* with 1 King, 1 Archer, 1 Warrior and 1 Healer.

Agent 1	Agent 2	Agent 1	Agent 2
LSI	**Rule-based**	45.0%(1.6)	**55.0%(1.6)**
Naive MCTS	Rule-based	**56.8%(1.4)**	43.2%(1.4)
MCTS	Rule-based	51.8%(1.6)	48.2%(1.6)
MCTS$_u$	Rule-based	**61.0%(1.0)**	39.0%(1.0)
Elastic MCTS$_u$	Rule-based	**57.8%(0.9)**	42.2%(0.8)
RG MCTS$_u$	Rule-based	**60.4%(0.9)**	39.6%(0.9)
LSI	MCTS	49.2%(1.1)	50.8%(1.1)
Naive MCTS	MCTS	**55.0%(2.1)**	45.0%(2.1)
MCTS$_u$	MCTS	**58.6%(1.6)**	41.4%(1.6)
Elastic MCTS$_u$	MCTS	**61.8%(1.6)**	38.2%(1.6)
RG MCTS$_u$	MCTS	**60.0%(1.9)**	40.0%(1.9)
LSI	**MCTS$_u$**	41.2%(1.6)	**58.8%(1.6)**
Naive MCTS	**MCTS$_u$**	47.6%(2.2)	**52.4%(2.2)**
Elastic MCTS$_u$	MCTS$_u$	**52.2%(1.9)**	46.4%(1.8)
RG MCTS$_u$	**MCTS$_u$**	47.4%(2.5)	**52.6%(2.5)**
LSI	**RG MCTS$_u$**	39.2%(1.5)	**60.8%(1.5)**
Naive MCTS	RG MCTS$_u$	48.0%(2.4)	52.0%(2.4)
Elastic MCTS$_u$	RG MCTS$_u$	**52.2%(1.9)**	46.4%(1.8)
LSI	**Elastic MCTS$_u$**	36.0%(1.6)	**64.0%(1.6)**
Naive MCTS	**Elastic MCTS$_u$**	41.0%(1.0)	**59.0%(1.0)**
Naive MCTS	LSI	**61.0%(1.4)**	39.0%(1.4)

We note that RG MCTS$_u$ shows competitive results compared to MCTS$_u$. While it might be the smaller tree size that helps, another possible reason is that nodes randomly grouped are from the same depth, where most of the nodes are controlling the same unit at the same horizon. Grouping these nodes benefits from their similar state-action distribution at an early stage (also note that α_{ABS} is also used for RG MCTS$_u$). In the results of games playing against MCTS$_u$, Elastic MCTS$_u$ shows higher win rates than its opponent while RG MCTS$_u$ shows similar or lower win rates. LSI shows lower win rates than MCTS$_u$while the performance of Naive MCTS is competitive to MCTS$_u$. In all army compositions, the Elastic MCTS$_u$ agent shows a consistently higher win rate than all its opponents, which shows the performance improvement brought by our proposed method. The usefulness of approximate MDP homomorphism can be verified by results of Elastic MCTS$_u$ playing against RG MCTS$_u$, where Elastic MCTS$_u$ outperforms RG MCTS$_u$ in all army compositions.

When the search complexity increases as the number of units grows from 4 units to 7 units and 10 units, the proposed method shows its consistency in outperforming all other agents. It is worth noting that the Elastic MCTS$_u$ performs better with more units (see results of Elastic MCTS$_u$ playing against the MCTS agent), which suggests a strong ability for our method to scale appropriately when using more units.

To evaluate the performance with different layouts and army compositions, we select the smallest 30 maps from [33], where we test 5 army compositions (see Table V) for each map. We evaluate the performance of Elastic MCTS$_u$ playing

TABLE III: Win rates with standard errors for the game *Kill The King* with 1 King, 2 Archers, 2 Warriors and 2 Healers.

Agent 1	Agent 2	Agent 1	Agent 2
LSI	Rule-based	49.6%(1.2)	50.4%(1.2)
Naive MCTS	Rule-based	**71.8%(1.7)**	28.2%(1.7)
MCTS	Rule-based	**54.4%(2.2)**	45.6%(2.2)
MCTS$_u$	Rule-based	**61.0%(1.0)**	39.0%(1.0)
Elastic MCTS$_u$	Rule-based	**74.0%(0.5)**	26.0%(0.5)
RG MCTS$_u$	Rule-based	**73.0%(0.8)**	27.0%(0.8)
LSI	**MCTS**	45.8%(2.0)	**54.2%(2.0)**
Naive MCTS	MCTS	**57.2%(0.7)**	42.8%(0.7)
MCTS$_u$	MCTS	**62.2%(0.8)**	37.4%(0.8)
Elastic MCTS$_u$	MCTS	**65.6%(1.6)**	34.2%(1.7)
RG MCTS$_u$	MCTS	**62.6%(1.4)**	37.4%(1.4)
LSI	**MCTS$_u$**	37.4%(0.6)	**62.6%(0.6)**
Naive MCTS	MCTS$_u$	51.6%(2.1)	48.2%(2.1)
Elastic MCTS$_u$	MCTS$_u$	**52.4%(2.0)**	44.2%(1.8)
RG MCTS$_u$	MCTS$_u$	50.8%(1.4)	48.6%(1.5)
LSI	**RG MCTS$_u$**	32.2%(0.9)	**67.8%(0.9)**
Naive MCTS	**RG MCTS$_u$**	42.8%(1.7)	**57.2%(1.7)**
Elastic MCTS$_u$	RG MCTS$_u$	**52.4%(2.0)**	44.2%(1.8)
LSI	**Elastic MCTS$_u$**	35.4%(1.4)	**64.6%(1.4)**
Naive MCTS	**Elastic MCTS$_u$**	47.4%(1.8)	**52.2%(1.7)**
Naive MCTS	LSI	**63.2%(1.5)**	36.8%(1.5)

TABLE IV: Win rates with standard errors for the game *Kill The King* with 1 King, 3 Archers, 3 Warriors and 3 Healers.

Agent 1	Agent 2	Agent 1	Agent 2
LSI	**Rule-based**	43.6%(1.3)	**56.4%(1.3)**
Naive MCTS	**Rule-based**	68.4%(1.0)	**31.6%(1.0)**
MCTS	**Rule-based**	45.8%(1.6)	**54.2%(1.6)**
MCTS$_u$	Rule-based	**65.6%(2.6)**	34.4%(2.6)
Elastic MCTS$_u$	Rule-based	**70.2%(0.7)**	29.6%(0.7)
RG MCTS$_u$	Rule-based	**68.6%(0.5)**	31.4%(0.5)
LSI	MCTS	51.2%(2.1)	48.8%(2.1)
Naive MCTS	MCTS	**65.0%(0.5)**	34.8%(0.5)
MCTS$_u$	MCTS	**62.8%(0.6)**	35.2%(1.0)
Elastic MCTS$_u$	MCTS	**70.0%(1.7)**	27.8%(1.2)
RG MCTS$_u$	MCTS	**65.6%(1.2)**	34.4%(1.2)
LSI	**MCTS$_u$**	34.6%(1.9)	**65.4%(1.9)**
Naive MCTS	**MCTS$_u$**	49.2%(1.7)	**50.0%(1.5)**
Elastic MCTS$_u$	MCTS$_u$	**50.2%(1.3)**	41.6%(1.7)
RG MCTS$_u$	MCTS$_u$	49.6%(1.1)	50.2%(1.1)
LSI	**RG MCTS$_u$**	32.0%(1.3)	**68.0%(1.3)**
Naive MCTS	**RG MCTS$_u$**	47.2%(1.1)	**52.6%(1.0)**
Elastic MCTS$_u$	RG MCTS$_u$	**50.2%(1.3)**	41.6%(1.7)
LSI	**Elastic MCTS$_u$**	31.4%(1.1)	**68.6%(1.1)**
Naive MCTS	**Elastic MCTS$_u$**	44.0%(2.1)	**55.0%(2.0)**
Naive MCTS	LSI	**63.4%(1.5)**	36.6%(1.5)

against MCTS$_u$. Table V shows the average win rates for both agents with their corresponding standard error between 5 random seeds. In 4 of the 5 army compositions, Elastic MCTS$_u$ outperforms MCTS$_u$ by large margins (between 7.7% and 14.4%). These results show that the proposed method improves the agent performance with state abstraction and its improvements are consistent in different army compositions and game maps. Specifically, in 4 of the 5 army compositions where Elastic MCTS$_u$ performs better are of large unit sizes. Similarly, results show that Elastic MCTS$_u$ obtains win rate improvements in *K2W2A2H* and *K3W3A3H* in comparison with *K1W1A1H* (see Tables III, IV and II, respectively).

TABLE V: Win rates with standard errors for games from 30 different layouts. The army composition indicates the number of warriors (*X*W), archers (*X*A), healers (*X*H) and 1 king (K).

Army Composition	Elastic MCTS$_u$	MCTS$_u$
K3H	41.0%(1.1)	**43.6%**(1.4)
K3W3A3H	**41.7%**(3.2)	34.0%(1.7)
K10A	**44.7%**(2.4)	30.3%(0.9)
K10W	**42.7%**(1.2)	29.0%(2.0)
K5W5A	**41.3%**(1.7)	31.7%(1.1)

Performance in Push Them All: The results for *PTA* are shown in Table VI. When playing against the rule-based agent, the MCTS agent shows a similar win rate as the rule-based ones, while the remaining agents outperform rule-based agents by large margins. Notably, LSI and Naive MCTS have higher win rates than agents with unit ordering, among which Elastic MCTS$_u$ shows the highest (9.4 higher than RG MCTS$_u$) win rate. When playing against MCTS, all gents based on MCTS$_u$, LSI and Naive MCTS outperform their opponents. Among these agents, LSI reaches the highest win rate (96.2%). RG MCTS$_u$ also shows a higher win rate than the MCTS$_u$ by a large margin. Next, by comparing Elastic MCTS$_u$ and RG MCTS$_u$ directly against MCTS$_u$, we observe a consistently high win rate of Elastic MCTS$_u$ (80.4%) while RG MCTS$_u$ has a lower win rate (66.8%). In the comparison between Elastic MCTS$_u$ and RG MCTS$_u$, Elastic MCTS$_u$ shows a higher win rate with a large difference of 20.6.

Both LSI and Naive MCTS show consistently higher win rates than other agents in this game, where LSI agent shows superior performance. These two agents benefit from evaluating action combinations rather than unit actions in all our MCTS agents which is important for concurrent movement. In all agents based on MCTS$_u$, Elastic MCTS$_u$has the highest win rates. RG MCTS$_u$ with abstraction threshold also shows improvements but is much weaker than Elastic MCTS$_u$. The results of these two agents confirm that grouping nodes controlling same-type units can benefit the performance where the automatic abstraction outperforms the random grouping.

Performance in Two Kingdoms: Table VII shows win rates for the game *TK*. When playing against the rule-based agent, MCTS gains a low win rate of 11.8%, while LSI obtains a win rate of 68.6%. All Naive MCTS, MCTS$_u$, Elastic MCTS$_u$ and RG MCTS$_u$ agents show a much stronger performance, with win rates of at least 85%. MCTS$_u$ shows the highest win rate of 91.0%. Elastic MCTS$_u$ has a win rate 90.2% comparable to MCTS$_u$ and higher than that of RG MCTS$_u$ (85.6%). All agents (except rule-based) outperform MCTS by large margins. When playing against MCTS$_u$, Elastic MCTS$_u$ outperforms MCTS$_u$with a win rate of 54.8%, while LSI, Naive MCTS, and RG MCTS$_u$ perform weaker than MCTS$_u$ with win rates 21.2%, 34.6% and 41.4% in win rates, respectively. When Elastic MCTS$_u$ plays against RG MCTS$_u$, Elastic MCTS$_u$ outperforms the latter with a win rate of 58.2, while LSI and Naive MCTS obtain low win rates in playing against both RG MCTS$_u$ and Elastic MCTS$_u$. Elastic MCTS$_u$ outperforms all our baseline algorithms.

TABLE VI: Win rates and standard errors for *Push Them All*

Agent 1	Agent 2	Agent 1	Agent 2
LSI	Rule-based	**97.6%(0.4)**	2.4%(0.4)
Naive MCTS	Rule-based	**85.8%(1.3)**	14.2%(1.3)
MCTS	Rule-based	48.6%(3.2)	51.2%(3.2)
MCTS$_u$	Rule-based	**65.8%(2.0)**	34.0%(2.1)
Elastic MCTS$_u$	Rule-based	**80.2%(1.3)**	19.8%(1.3)
RG MCTS$_u$	Rule-based	**70.8%(2.0)**	29.2%(2.0)
LSI	MCTS	**96.2%(1.0)**	3.8%(1.0)
Naive MCTS	MCTS	**86.0%(1.4)**	12.2%(1.3)
MCTS$_u$	MCTS	**60.2%(2.4)**	37.0%(2.0)
Elastic MCTS$_u$	MCTS	**87.2%(1.0)**	12.4%(0.8)
RG MCTS$_u$	MCTS	**82.6%(1.5)**	16.4%(1.8)
LSI	MCTS$_u$	**93.8%(0.6)**	5.6%(0.7)
Naive MCTS	MCTS$_u$	**86.0%(0.9)**	13.2%(1.1)
Elastic MCTS$_u$	MCTS$_u$	**80.4%(1.1)**	18.8%(1.2)
RG MCTS$_u$	MCTS$_u$	**66.8%(2.2)**	32.2%(2.2)
LSI	RG MCTS$_u$	**87.4%(1.6)**	12.0%(1.8)
Naive MCTS	RG MCTS$_u$	**67.0%(2.4)**	31.6%(2.2)
Elastic MCTS$_u$	RG MCTS$_u$	**59.8%(1.7)**	39.2%(1.8)
LSI	Elastic MCTS$_u$	**71.4%(1.4)**	27.8%(1.3)
Naive MCTS	Elastic MCTS$_u$	**55.2%(1.1)**	43.4%(1.2)
Naive MCTS	**LSI**	36.4%(1.8)	**63.6%(1.8)**

TABLE VII: Win rates and standard errors for *Two Kingdoms*

Agent 1	Agent 2	Agent 1	Agent 2
LSI	Rule-based	**68.6%(1.4)**	31.4%(1.4)
Naive MCTS	Rule-based	**85.0%(1.0)**	14.4%(1.3)
MCTS	**Rule-based**	11.8%(1.7)	**83.4%(1.6)**
MCTS$_u$	Rule-based	**91.0%(1.5)**	6.0%(1.2)
Elastic MCTS$_u$	Rule-based	**90.2%(2.0)**	8.6%(1.7)
RG MCTS$_u$	Rule-based	**85.6%(1.6)**	14.4%(1.6)
LSI	MCTS	**82.4%(1.3)**	17.6%(1.3)
Naive MCTS	MCTS	**90.8%(1.0)**	9.2%(1.0)
MCTS$_u$	MCTS	**97.2%(0.3)**	2.8%(0.3)
Elastic MCTS$_u$	MCTS	**95.2%(1.3)**	4.8%(1.3)
RG MCTS$_u$	MCTS	**92.0%(0.6)**	8.0%(0.6)
LSI	**MCTS$_u$**	21.2%(0.8)	**78.8%(0.8)**
Naive MCTS	**MCTS$_u$**	34.6%(1.9)	**65.4%(1.9)**
Elastic MCTS$_u$	MCTS$_u$	**54.8%(2.8)**	45.0%(2.8)
RG MCTS$_u$	**MCTS$_u$**	41.4%(1.7)	**58.6%(1.7)**
LSI	**RG MCTS$_u$**	22.6%(2.1)	**77.4%(0.5)**
Naive MCTS	**RG MCTS$_u$**	39.0%(0.7)	**61.0%(0.7)**
Elastic MCTS$_u$	RG MCTS$_u$	**58.2%(1.5)**	41.6%(1.6)
LSI	**Elastic MCTS$_u$**	16.6%(1.4)	**83.4%(1.4)**
Naive MCTS	**Elastic MCTS$_u$**	30.8%(0.5)	**69.2%(0.5)**
Naive MCTS	LSI	**66.6%(2.9)**	33.4%(2.9)

We evaluate Elastic MCTS$_u$ in a strategy game with more unit types and more complex tasks. This method shows a consistent win rate improvement. However, RG MCTS$_u$, which randomly groups nodes in the same depth, performs badly in this game because different units are spawned in different branches: there is a low probability for grouping nodes that control the same type of units.

C. Influence of Abstraction Threshold

To investigate the influence of the iteration threshold α_{ABS}, we pitch Elastic MCTS$_u$ against MCTS$_u$, with similar settings as in the *KTK* experiments (Table II, III and IV), but different values of the α_{ABS} are explored. This experiment explores the values of this parameter to observe the effect of using state abstraction during different proportions of MCTS iterations. We evaluate the average win rates of these two agents with different proportions $\alpha_{ABS} \in \{0\%, 25\%, 50\%, 75\%, 100\%\}$ are set for Elastic MCTS$_u$. The α_{ABS} set to 0% makes the Elastic MCTS$_u$ behave like MCTS$_u$. When it is 100%, all MCTS iterations are run under the state abstraction, where the recommendation policy picks actions based on group statistics. This is similar to its usage in previous work [9], [11], [13].

TABLE VIII: Win rates with standard errors of Elastic MCTS$_u$ vs MCTS$_u$. The proportion column indicates at which % of the search the state abstraction is abandoned.

Composition	Prop.	Elastic MCTS$_u$	MCTS$_u$	Diff.
KWAH	0%	47.0%(2.3)	52.8%(2.3)	−5.8
	25%	48.6%(1.1)	51.4%(1.1)	−2.8
	50%	**53.2%(1.7)**	**46.2%(1.7)**	**7.0**
	75%	51.4%(0.7)	46.6%(0.5)	4.8
	100%	49.8%(1.9)	46.6%(1.6)	3.2
K2W2A2H	0%	52.4%(2.6)	46.8%(2.6)	5.6
	25%	49.8%(1.5)	48.6%(1.7)	1.2
	50%	51.0%(1.3)	45.6%(1.4)	5.4
	75%	**51.8%(1.4)**	**43.4%(1.3)**	**8.4**
	100%	47.4%(1.2)	45.1%(1.2)	2.3
K3W3A3H	0%	48.8%(1.6)	48.6%(1.7)	0.2
	25%	**53.4%(1.5)**	**44.0%(1.1)**	**9.4**
	50%	48.4%(0.8)	45.2%(1.2)	3.2
	75%	44.6%(1.6)	46.6%(1.9)	−2.0
	100%	48.6%(0.8)	40.6%(1.0)	8.0

As is shown in Table VIII, the performance of the Elastic MCTS$_u$ agent differs in different abstraction threshold values. We highlight win rates that have the largest difference from the opponents' win rates. Note that the different hyper-parameters (tuned by NTBEA) used by these two agents result in their different win rates with $\alpha_{ABS} = 0\%$, where both agents do not use state abstraction. In each army composition, there seems to be a sweet spot on the value of abstraction threshold: performance is best when using the state abstraction during the first 25% and 75% iterations of Elastic MCTS$_u$, obtaining lower winning rates when closer to 100%. We note that, although the highest win rate of $K2W2A2H$ shows up with $\alpha_{ABS} = 0\%$, the difference in winning rate makes $\alpha_{ABS} = 75\%$ the better choice as it reduces games lost.

The best values found for α_{ABS} vary with the environment settings. In all cases, the best performance is obtained using an abstraction threshold $< 100\%$. Our interpretation of this phenomenon is that nodes grouped together sharing the same statistics until the end of the decision process leads to a sub-optimal recommendation policy, as the optimal action might be grouped together with other sub-optimal actions which makes it indistinguishable. However, abandoning the abstraction at an intermediate iteration during the search lands better results in all army compositions. This is likely because it allows the search to further explore nodes within the same group and refine the final action sequence.

D. Compression Rate

To visualize the difference between the original and the abstracted search space, we define a compression rate for an

abstracted MCTS tree as N_{tree}/N_{abs_tree}, where N_{tree} is the number of grounds nodes generated by MCTS and N_{abs_tree} is the number of node groups generated by the abstraction. The compression ratio compares the size of the same tree with and without abstraction. Note that a lower group number for the same tree means the tree size is reduced further. We evaluate compression rates for all games used in this paper. For each game, we run Elastic MCTS$_u$ in 10 games with $\alpha_{ABS} = 100\%$ and record the compression rates in different MCTS iterations , which are visualized in Fig. 3 The vertical line in each figure shows the α_{ABS} resultant from the tuning done by NTBEA.

For *KTK*, we evaluate the compression rates in three army compositions from 4 to 10 units. As shown in Fig. 3a, all the compression rates show a moderate increase over time but at different increasing rates. The *1K2W2A2H* composition has the highest increasing rate, followed by the *1K3W3A3H*. *1K1W1A1H* composition grows close to *1K3W3A3H* but shows a slightly slower increasing after 220 iterations. With the tuned α_{ABS}, the sizes of trees under abstraction (with $\alpha_{ABS} \times B = 10 \times 20$) are 5-7 times smaller than the original tree size. If the abstraction is used until the end of MCTS, we observe the tree sizes are reduced with factors from 7 to 10, while the performances are competitive to MCTS$_u$ (See Table VIII with 100% iterations using abstraction). The highest compression rate is around 10 for *K3W3A3H*.

In *PTA*, we observe a lower compression rate compared to *KTK*. In the performance evaluation for *PTA*, the α_{ABS} is set as 8 and the compression rate is around 1.7. The lower compression rate is possibly related to the smaller action space and fewer unit types in this game. Combined with the results in Table VI, it suggests that this degree of compression correlates with a performance improvement. Our visualization for RG MCTS$_u$ agent shows a smaller tree size than the tree under approximate MDP homomorphism.

For *TK*, the overall compression rate of Elastic MCTS$_u$ is around 3, and the compression rate in the performance evaluation is 3.0 (Table VII). In *TK*, it has a complex heuristic function considering different tasks, making it challenging to select approximate thresholds (see Eq. 2) for state abstraction. This also trades off node grouping between different units. For the RG MCTS$_u$ agent, the compression rate is overall smaller.

We have shown that state abstraction reduces the tree size, but another question to answer is: *Is the proposed method computationally efficient compared to MCTS$_u$?* We found that even though the Elastic MCTS$_u$ requires more computation – computing the approximate errors and grouping nodes, the total computation time is close to the MCTS$_u$ agent. With a large budget of $30,000$ forward model calls for *KTK* game, the average decision times are 667 ± 13 and 685 ± 13 ms for MCTS$_u$ and Elastic MCTS$_u$, respectively.

In conclusion, we analyzed the tree size of MCTS$_u$, Elastic MCTS$_u$, and RG MCTS$_u$. We observe an increasing compression rate over time, with Elastic MCTS$_u$ and RG MCTS$_u$ reduced the tree size. The proposed method, Elastic MCTS$_u$ reaches the best performance with compression rates among $1.6 - 7$ in different games. With a smaller tree size, we found the difference in computation time between Elastic MCTS$_u$ and MCTS$_u$ is within 20ms with a large budget.

VII. Conclusion and Future Work

In this paper, we propose a new algorithm, Elastic MCTS, which is a variant of MCTS that groups tree nodes using state abstraction based on MDP Homomorphism. This work aims to develop automatic state abstraction methods for general strategy game playing. Our work is inspired by state abstractions studied in planning domains, but it is adapted for complex search and action spaces. The proposed ungrouping mechanism that splits the state abstraction back to the ground states has been shown essential for search performance. We extend our previous work [17] from one game to three games to evaluate the scalability of the proposed method. A new baseline, the random-grouping agent, has been used to verify the usefulness of MDP homomorphism.

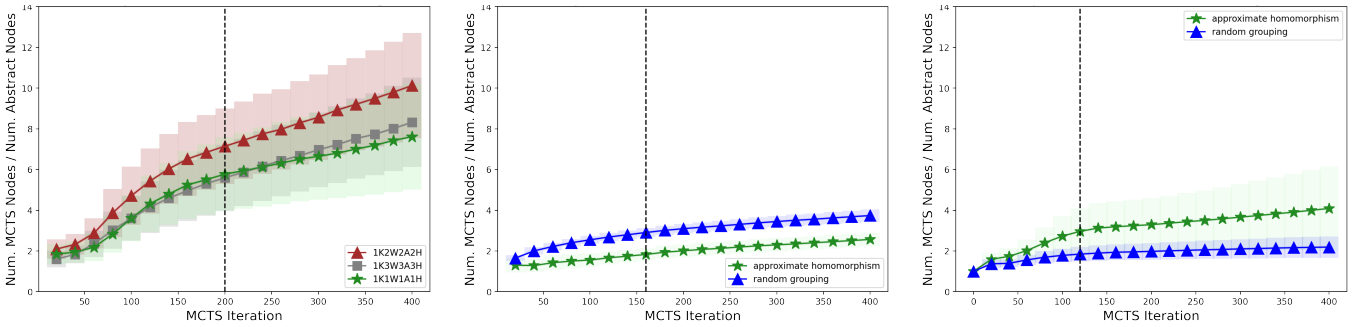
Our experiments show that the proposed method outperforms MCTS and a rule-based agent in all three turn-based strategy games. By observing that a random grouping agent fails to bring stable improvement in performance (sometimes harming it), MDP homomorphism is confirmed to be essential for performance improvement. By analyzing the tree size with/without state abstraction, we also observe a reduction of the tree size by a factor of 10 and a considerable performance improvement when using state abstraction during a proportion of the search, as long as we expand the search tree to its ground state before the recommendation policy selects an action.

The results shown in this paper have been obtained in three turn-based games, but we hypothesize that the proposed methods may also benefit other real-time or turn-based strategy games. Due to the increased number of units controlled in games such as Starcraft, the search problem becomes more challenging. Nevertheless, we believe that the observed efficiency improvements may transfer well. This is an immediate case for future work, where the applicability and scalability of Elastic MCTS will be investigated in more complex games.

Having observed the performance improvement in gameplaying, a natural question to ask is: *what else can state abstraction do?*. This question leads to several research directions. For example, whether different gameplay styles be obtained by using state abstractions or if, on the contrary, the reduced tree prevents the agent from that goal. Recent works on quality-diversity methods applied to strategy games [34] could be explored in conjunction with Elastic MCTS. Additionally, action space reduction can be applied on Elastic MCTS to see the potential synergies, e.g., with portfolio methods [2]. Finally, automatic state abstraction applying to strategy games is still a new direction and different mechanisms for state abstraction (e.g. [12]) could be an alternative to approximate MDP homomorphism. In fact, considering the long horizon in complex strategy games, it's possible to divide the game into different stages. Each stage may require different state abstractions, which lies an interesting line of research ahead to discover context-aware versions of Elastic MCTS.

Acknowledgments

For the purpose of open access, the author(s) has applied a Creative Commons Attribution (CC BY) license to any Accepted Manuscript version arising. This work is supported by UK EPSRC grant EP/T008962/1.



(a) Kill The King (*KTK*) (b) Push Them All (*PTA*) (c) Two Kingdoms (*TK*)

Fig. 3: Compression rates for each tested game including standard errors from 10 game plays.

REFERENCES

[1] D. Churchill and M. Buro, "Build order optimization in starcraft," in *Seventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2011.

[2] ——, "Portfolio greedy search and simulation for large-scale combat in starcraft," in *2013 IEEE Conference on Computational Inteligence in Games (CIG)*. IEEE, 2013, pp. 1–8.

[3] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-carlo tree search: A new framework for game ai," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 4, no. 1, 2008, pp. 216–217.

[4] G. Konidaris, "On the necessity of abstraction," *Current opinion in behavioral sciences*, vol. 29, pp. 1–7, 2019.

[5] A. Uriarte and S. Ontanón, "Game-tree search over high-level game states in rts games," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 10, no. 1, 2014, pp. 73–79.

[6] ——, "High-level representations for game-tree search in rts games," in *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2014.

[7] N. A. Barriga, M. Stanescu, and M. Buro, "Puppet search: Enhancing scripted behavior by look-ahead search with applications to real-time strategy games," in *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2015.

[8] ——, "Combining strategic learning with tactical search in real-time strategy games," in *Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2017.

[9] N. Jiang, S. Singh, and R. Lewis, "Improving uct planning via approximate homomorphisms," in *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, 2014, pp. 1289–1296.

[10] J. Hostetler, A. Fern, and T. Dietterich, "State aggregation in monte carlo tree search," in *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, C. E. Brodley and P. Stone, Eds. AAAI Press, 2014, pp. 2446–2452.

[11] A. Anand, A. Grover, M. Mausam, and P. Singla, "Asap-uct: Abstraction of state-action pairs in uct," in *Proceedings of the 24th International Conference on Artificial Intelligence*, ser. IJCAI'15. AAAI Press, 2015, p. 1509–1515.

[12] J. Hostetler, A. Fern, and T. G. Dietterich, "Progressive abstraction refinement for sparse sampling." in *UAI*, 2015, pp. 365–374.

[13] A. Anand, R. Noothigattu, P. Singla *et al.*, "Oga-uct: On-the-go abstractions in uct," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 26, 2016.

[14] J. Hostetler, A. Fern, and T. Dietterich, "Sample-based tree search with fixed and adaptive state abstractions," *Journal of Artificial Intelligence Research*, vol. 60, pp. 717–777, 2017.

[15] S. Sokota, C. Y. Ho, Z. Ahmad, and J. Z. Kolter, "Monte carlo tree search with iteratively refining state abstractions," *Advances in Neural Information Processing Systems*, vol. 34, pp. 18 698–18 709, 2021.

[16] B. Ravindran and A. G. Barto, "Approximate homomorphisms: A framework for non-exact minimization in markov decision processes," in *Intl. Conference on Knowledge-Based Computer Systems*, 2004.

[17] L. Xu, J. Hurtado-Grueso, D. Jeurissen, D. P. Liebana, and A. Dockhorn, "Elastic monte carlo tree search with state abstraction for strategy game playing," in *2022 IEEE Conference on Games (CoG)*. IEEE Press, 2022, p. 369–376. [Online]. Available: https://doi.org/10.1109/CoG51982.2022.9893587

[18] A. Shleyfman, A. Komenda, and C. Domshlak, "On combinatorial actions and cmabs with linear side information," in *ECAI 2014*. IOS Press, 2014, pp. 825–830.

[19] S. Ontanón, "The combinatorial multi-armed bandit problem and its application to real-time strategy games," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 9, no. 1, 2013, pp. 58–64.

[20] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European Conf. on machine learning*. Springer, 2006, pp. 282–293.

[21] H. Finnsson, "Generalized monte-carlo tree search extensions for general game playing," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 26, no. 1, 2012, pp. 1550–1556.

[22] R. Lorentz, "Early playout termination in mcts," in *Advances in Computer Games: 14th International Conference, ACG 2015, Leiden, The Netherlands, July 1-3, 2015, Revised Selected Papers 14*. Springer, 2015, pp. 12–19.

[23] J.-T. Saito, M. H. Winands, J. W. Uiterwijk, and H. J. Van Den Herik, "Grouping nodes for monte-carlo tree search," in *Computer Games Workshop*, 2007, pp. 276–283.

[24] M. Chung, M. Buro, and J. Schaeffer, "Monte carlo planning in rts games," in *IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2005, pp. 1–8.

[25] G. Synnaeve and P. Bessiere, "A bayesian tactician," in *Computer Games Workshop at ECAI 2012*, 2012, pp. pp–114.

[26] A. Dockhorn, D. Perez-Liebana *et al.*, "Game state and action abstracting monte carlo tree search for general strategy game-playing," in *2021 IEEE Conference on Games (CoG)*. IEEE, 2021, pp. 1–8.

[27] G. Van Eyck and M. Müller, "Revisiting move groups in monte-carlo tree search," in *Advances in Computer Games: 13th International Conference, ACG 2011, Tilburg, The Netherlands, November 20-22, 2011, Revised Selected Papers 13*. Springer, 2012, pp. 13–23.

[28] A. Dockhorn, J. H. Grueso, D. Jeurissen, and D. P. Liebana, "Stratega: A general strategy games framework." in *AIIDE Workshops*, 2020.

[29] A. Dockhorn, D. Perez-Liebana *et al.*, "Portfolio search and optimization for general strategy game-playing," in *2021 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2021, pp. 2085–2092.

[30] G.-J. Roelofs, "Pitfalls and solutions when using monte carlo tree search for strategy and tactical games," in *Game AI Pro 3*. AK Peters/CRC Press, 2017, pp. 343–354.

[31] N. Sato and K. Ikeda, "Three types of forward pruning techniques to apply the alpha beta algorithm to turn-based strategy games," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2016, pp. 1–8.

[32] S. M. Lucas, J. Liu, and D. Perez-Liebana, "The n-tuple bandit evolutionary algorithm for game agent optimisation," in *2018 IEEE Congress on Evolutionary Computation (CEC)*, 2018, pp. 1–9.

[33] N. R. Sturtevant, "Benchmarks for grid-based pathfinding," *IEEE Trans. on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 144–148, 2012.

[34] D. Perez-Liebana, C. Guerrero-Romero *et al.*, "Generating diverse and competitive play-styles for strategy games," in *2021 IEEE Conference on Games (CoG)*. IEEE, 2021, pp. 1–8.