Strategy Identification and Clustering in StarCraft II Trajectories

1 StarCraft II as a Domain

StarCraft II is a real-time strategy game developed by Blizzard Entertainment that can be viewed as a challenging *multi-agent reinforcement learning* (MARL) domain. In StarCraft, players control factions consisting of numerous units and structures, engaging in complex battles and resource management.

As a MARLdomain, StarCraft offers several characteristics that make it appealing for research and development in reinforcement learning:

Partial observability. Each player has limited visibility over the game state and is often unaware of the opponent's actions and intentions. Agents must use their available information and make decisions based on incomplete knowledge.

Large action space. StarCraft provides a vast set of possible actions for each agent. Units can move, attack, use special abilities, and construct buildings, leading to a high-dimensional and continuous action space.

Heterogeneous agents. Different units in StarCraft possess unique capabilities and roles. For instance, some units excel at ranged attacks, while others specialize in healing or reconnaissance. Agents must learn to coordinate and utilize their units effectively to achieve victory.

Multi-scale decision-making. The game operates on multiple spatial and temporal scales. Agents need to make decisions at the unit level, group level, and global level simultaneously, managing both micro and macro aspects of gameplay.

Stochasticity. StarCraft introduces uncertainty through random events, such as misses in combat or random maps. Agents must adapt their strategies to handle probabilistic outcomes.

From a multi-agent reinforcement learning (MARL) perspective, the mutli-scale decision-making can be split into two: micro-management and macro-management. Micro-management involves the actions taken by individual units or small groups of units during battles. On the other hand, macro-management in StarCraft refers to the higher-level strategic decisions that shape the overall game plan.

The focus of this work, as well as the environment used is on the micro-scale. While the macro-scale offers a great variety of strategies composed of multiple parts and sub-strategies, it is too complex for the scope of this work. Therefore, we focus on selected battle scenarios, and analyze the different strategies learned by MARLmodels.

1.1 Project Goal

While it does seem obvious that different scenarios need different strategies, do they all differ greatly? Do some have something in common which can be identified and explored?

These questions are the focal point of the project. Therefore, the project's goal is to identify, cluster, and characterize different strategies learned by MARL models in the StarCraft II environment in battle-focused micro-management oriented scenarios.

2 Environment Setup and Data Extraction

This work utilizes two main repositories in order to produce the trajectories used in later parts of the work: SMAC[1] [Samvelyan *et al.*, 2019] and PyMARL2[2] [Hu *et al.*, 2021].

2.1 Environment Setup

The environment chosen was SMAC 1 and not SMAC 2 due to previous studential work on the topic as well as ease of running using pymarl2 compared to rllib and the original pymarl which we were not able to run.

The versions of packages and the environment itself are very important for running. The codebase includes two files, which include all the necessary conda packages and the pip packages alongside their respective versions.

As of writing this report, July 3 2023, the following commands were used (in order):

```
conda create -n pymarl_3 python=3.8
 ↪  cudatoolkit=11.1 -y
conda activate pymarl_3
conda install pytorch torchvision
 ↪  torchaudio -c pytorch-lts -c nvidia -y
pip install sacred numpy scipy gym==0.10.8
 ↪  matplotlib seaborn pyyaml==5.3.1 pygame
 ↪  pytest probscale imageio snakeviz
 ↪  tensorboard-logger
pip install git+
 ↪  https://github.com/oxwhirl/smac.git
pip install protobuf
```

[1] https://github.com/oxwhirl/smac

[2] https://github.com/hijkzzz/pymarl2

Note that later we needed to degrade numpy to 1.23 from 1.24 because of issues with deprecation of bool.

2.2 Heuristic Agents

In the SMAC repository, under examples, there is a simple example of random agents. This allows for easy interfacing to the environment and figuring out important features it provides. One of those features is *heuristic_ai* which turns causes the system to use heuristic agents.

These agents do not learn their environment, but instead each agent searches for its nearest enemy, and shoots until one of them is dead. If the agent is still alive, it searches for the next target and so on until the fight is over.

2.3 Learning Agents

For the learning agents, we utilized QMIX [Rashid *et al.*, 2018] utilizing the implementation in pymarl2. The hyperparameters stayed the same, and we trained a separate model for each scenario and difficulty. The trajectories were extracted using a checkpoint of the model after roughly 10M steps.

The models were trained concurrently on the university's CPU cluster. Since the training time per scenario was between 12 to 72 hours, we did not explore the option of utilizing the GPU cluster as we preferred investing more time into analyzing the trajectories.

2.4 Running on the Cluster

The CPU cluster seems to lack GLIBC_2.18, therefore you need to build and install it yourself using the following commands (note the space in glibc-2.18... which was used to fit everything into the column):

```
curl -O http://ftp.gnu.org/gnu/glibc/glibc
 ↪  -2.18.tar.gz
tar zxf glibc-2.18.tar.gz
cd glibc-2.18/
mkdir build
cd build/
../configure --prefix=/usr
make -j2
# This line is to move libc.a to the lib of
 ↪  conda
mv libc.a $CONDA_PREFIX/lib/
# Then make sure to point to it every time
 ↪  you activate an environment (including
 ↪  in the slurm calls)
export LD_LIBRARY_PATH=$CONDA_PREFIX/lib/
```

2.5 Trajectory Extraction

By turning on the debug option of the environment, setting a correct formatter, and cleaning the logs afterwards, we were able to extract trajectories that include the initial state, and the state, observations, actions, and reward per step on the environment. While this work mostly concerns itself with the actions of the agents, the data of the trajectories remains intact including the ability to extract it from future logs.

Once a log is loaded, it turns into a list of trajectories (also called episodes), each in itself is a list of steps, and in each step we have the actions of each agent, which is what we take onwards to the data representation step.

	Map Name	**Win Rate**
Easy Hard	2s vs 1sc	100%
	2s3z	100%
	3s vs 5z	100%
	3s5z	100%
	8m	100%
	8m vs 9m	96%
	MMM	100%
Heuristic	2s vs 1sc	0%
	8m	80%

Table 1: Trajectory dataset

	Dead	Move East	Move North	Move South	Move West	Stop	Combined Attacks	Combined Heals
0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.4	0.0	0.0	0.0	0.3	0.3	0.0
4	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.9	0.1
6	0.0	0.6	0.0	0.1	0.0	0.2	0.0	0.1
7	0.0	0.0	0.0	0.0	0.0	0.0	0.9	0.1
8	0.0	0.2	0.0	0.0	0.0	0.7	0.0	0.1

Figure 1: Instead of treating the data as a sequence of actions for each agent ID, the percentage of agents that performed each type of action in each timestamp is represented. That way, a trajectory can be treated as a multivariate time series.

The final dataset is shown in Table 1 consists of 25 trajectories per map. Easy and Hard are combined as each map have the same win percentage, though easy and hard lost in different instances of the same map.

3 Method and Analysis

3.1 Trajectory Representation

The reinforcement learning algorithms, described in section 2.3 outputs a trajectory for each scenario. A trajectory is a sequence of the actions taken by the agent at each time step. The actions that the environment allows these algorithms to perform are: Move (east, west, north, south), Stop, Attack enemy ID, Heal agent ID and Dead.

Each trajectory forms an action vector that each agent did for each time-step. This type of representation highlights and links actions to specific agents, which leads to individual agent strategy analysis. The goal of our project was to try and learn about how the individual agents work as a team, and to analyze the strategy of all of the agents together. Therefore, a different representation was needed.

Therefore, we transformed the dataset in a way that allows to understand the group trajectory, while highlighting individual agent actions. In our chosen representation, as seen in Figure 1, each column is a possible action, and the values represent the percentage of agents that performed that action in a specific timestamp. This representation can also be interpreted as a multivariate time series, where each type of action serves as a feature or sensor. An example for the new dataset representation can be seen in Figure 2.

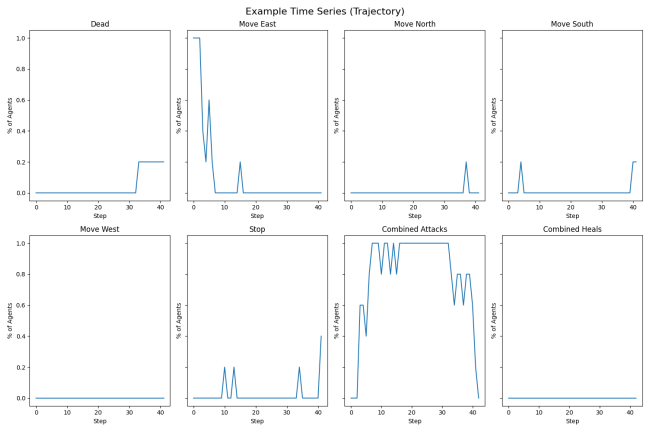

Figure 2: Dataset entries as a multivariate time series

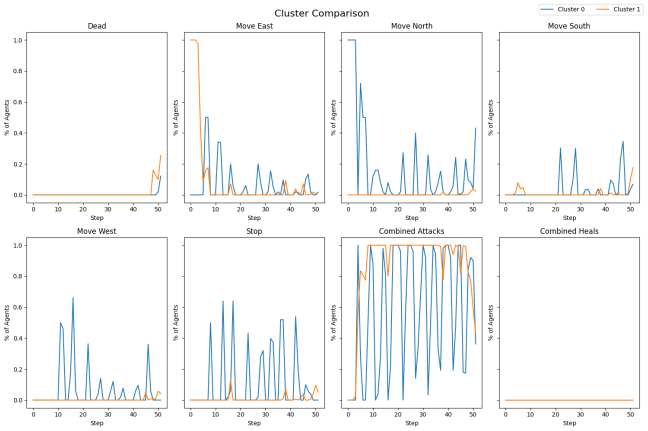

Figure 3: Cluster center comparison: $2s3z$ and $2s\_vs\_1sc$

3.2 Time Series Clustering

Similarly to tabular-data clustering, time series clustering is an unsupervised data mining technique used for organizing data entries (in this case: time series) into groups based on their similarity. The objective is to maximize data similarity within clusters and minimize it across clusters. In order to measure similarity between entries, a similarity measure needs to be defined. An example for similarity measure can be euclidean distance between each time-step for each feature. However, since our trajectories have varying length, largely determined by the complexity of the scenario, direct application of the euclidean distance is not feasible. This is because not all time-steps in one trajectory will have matching time-steps in another trajectory.

Therefore, in our project, we used dynamic time warping (DTW) as a similarity measure between entries. In general, DTW is a method that calculates an optimal match between two given time series, with certain rules:

1. Every index in the first time series must be match with one or more indices from the other time series, and vice versa.
2. The first (last) index from the first time series must be matched with the first (last) index from the other time series, but it does not have to be the only match.
3. The mapping of the indices from the first time series to the other time series must be monotonically increasing, and vice versa.

In other words, DTW seeks for temporal alignment between two time series of different lengths that will minimize the euclidean distance between the aligned series.

In order to index all time series, despite their different lengths, and in order to use DTW for similarity measure, we used Python tslearn library[3], and transformed the dataset into a time-series-dataset. Then, clustering of the multivariate time series can be done using a time series clustering algorithm, such as tslearn TimeSeriesKMeans.

As an example, we performed time series K means for two scenarios, $2s3z$ and $2s\_vs\_1sc$, for a total of 50 trajectories.

[3]https://tslearn.readthedocs.io/en/stable/index.html

For this example, we assumed that each scenario would result in a different strategy, and therefore we used time series K-means with $K = 2$. The purpose of this example is to demonstrate our method, which will be used later for the whole dataset. Figure 3 shows a comparison between the two cluster centers that were discovered. Since we cluster multivariate time series, the time series similarity is measured and summed across all features, and therefore the cluster centers can be represented as multivariate time series as well. In this example, agents of Cluster 1 rush to move east, where the enemy units are, and then stay in place and focus fire. In comparison, agents of Cluster 0 spread: they split into smaller groups that each moves to a different direction. Agents in Cluster 0 repeat a sequence of moving in the assigned direction and then firing. Therefore, Cluster 1 can be described as "Push" or "Attack nearest enemy", while Cluster 0 can be described as "Bait and Attack (Kiting)".

The main challenge in analyzing and categorizing cluster centers using the multivariate time-series representation can be challenging. It is hard to compare all differences between all features at the same time, especially when a larger number of clusters is used. Therefore, in the project, we suggest a new multivariate time series representation, called "Main Action Graph", which we will use to analyze the cluster centers. This representation aggregates all features in a multivariate time series into a univariate time series, by only showing the action that was done by the highest percentage of agents for each time-step. The same analysis and conclusions that were shown using the multivariate time series can be done using the Main Action Graph representation, as seen in Figure 4.

Since in many cases, an equal percentage of agents perform different kinds of actions, using the Main Action representation will not show the full strategy. Therefore, in Figure 5 all actions are stacked into one graph that represents the whole cluster center.

3.3 Finding the Optimal Number of Clusters

In this subsection we will start describing the results of our project regarding all trajectories of all maps, as described in Table 1. Our experiments contain 400 trajectories that belong to 7 different SMAC scenarios. Scenarios $8m$ and $2s\_vs\_1sc$

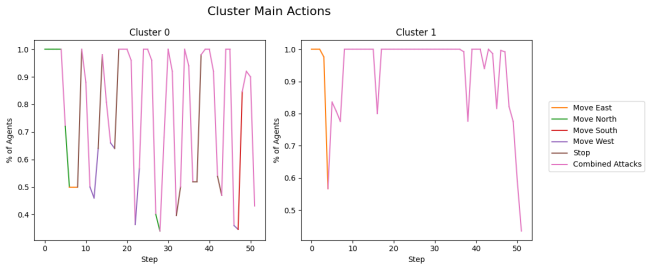

Figure 4: Main Action Graph representation

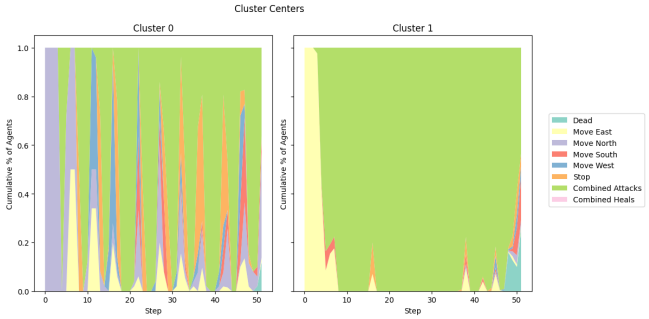

Figure 5: Action stack plot for cluster centers

were evaluated using QMIX against easy and hard mode, as well as using the heuristic option described in subsection 2.2. The other maps were evaluated using QMIX against easy and hard mode.

It is important to mention that in the original trajectories, an attack action is linked to an enemy ID, and a heal action is linked to an agent ID. Since different scenarios have a varying number of enemies and agents, using an attack action that is linked to a specific ID for clustering can result in grouping maps based on the number of enemies or agents in it. Not only it will lead to a biased clustering, but it will also lead to focusing on specific agent or enemy IDs, and therefore will not cluster based on group strategy. Therefore, all attack actions were summed into one "Combined Attacks" action, which does not specify the target of the attack. Similarly, all heal actions were summed into "Combined Heals".

In order to cluster using time series K-means, an optimal number of clusters K needs to be defined or discovered. The assumption that each scenario would be played using a different strategy can potentially lead to biased results. It is important to consider that multiple scenarios can be played using the same strategy, and therefore choosing K solely based on the number of scenarios might not accurately reflect the underlying patterns in the data. Therefore, we decided to use a data-driven approach using the elbow-method in order to determine the appropriate number of clusters for the analysis. The elbow-method is a heuristic to the optimal number of clusters. Figure 8 shows the inertia for each number of clusters, and the optimal value K is where the inertia decreases rapidly for each $k < K$ and slowly for $k > K$. According to Figure 8, the optimal number of clusters should be 4, 5 or 6.

Map Name	Difficulty	Cluster 0	Cluster 1	Cluster 2	Cluster 3	Cluster 4
2s3z	Easy	0%	100%	0%	0%	0%
	Hard	0%	100%	0%	0%	0%
2s_vs_1sc	Easy	100%	0%	0%	0%	0%
	Hard	100%	0%	0%	0%	0%
	Heuristic	0%	0%	0%	**100%**	0%
3s5z	Easy	0%	100%	0%	0%	0%
	Hard	0%	100%	0%	0%	0%
3s_vs_5z	Easy	0%	0%	**100%**	0%	0%
	Hard	0%	0%	0%	0%	**100%**
8m	Easy	0%	100%	0%	0%	0%
	Hard	0%	100%	0%	0%	0%
	Heuristic	0%	100%	0%	0%	0%
8m_vs_9m	Easy	0%	100%	0%	0%	0%
	Hard	0%	100%	0%	0%	0%
MMM	Easy	0%	100%	0%	0%	0%
	Hard	0%	100%	0%	0%	0%

Table 2: Scenario-Cluster Mapping

3.4 Results

Figure 6 shows the cluster centers for clustering all scenarios, using 5 clusters and the method described in section 3.2. In addition, Figure 7 shows the stacked action graph for each cluster center. Analyzing each cluster center points at the following strategies:

1. Cluster 0 - "Wait and Attack", or "Flanking" - Agents start by moving north - around the enemy units, and then mostly focus on attacking. At some steps the agents stop - it might be as they wait for the enemy units to move closer to them.
2. Cluster 1 - "Attack Nearest Enemy" - Agents move east, towards the enemy units, and then focus on attacking.
3. Cluster 2 - "Spreading" - Agents spread around the enemy units and attack from all directions. The stacked action graph shows that agents spread evenly across all directions, which shows that the model learned for the agent to act and plan as a group.
4. Cluster 3 - This is a very simple and straight forward strategy, which shows that for the larger part of the scenario, agents only attack and do nothing else.
5. Cluster 4 - "Spreading" - Agents spread around the enemy units and attack from all directions. The stacked action graph shows that agents spread evenly across all directions, which shows that the model learned for the agent to act and plan as a group.

Table 2 shows assignment of each scenario a cluster. According to the table, we can see that many scenarios resulted in a similar strategy: 2s3z, 3s5z, 8m, 8m_vs_9m, MMM. Some of these scenarios (2s3z-3s5z, 8m-8m_vs_9m) represent scenarios that use the same unit-types, with a varying number of units of each type. Therefore, It seems like what mostly affects the strategy is the unit-type that participate, and not the number of unit within the scenario, as long as some balance between sides is kept.

In addition, in most cases, no difference in strategies was detected when changing from fighting enemies in an easy or hard mode. The only scenario in which changing the enemy skill level resulted in a cluster change was scenario 3s_vs_5z, which was assigned to Cluster 2 and 4. Both clusters were marked as "Spreading", and it was hard to distinguish any meaningful difference between the two. The cluster centers

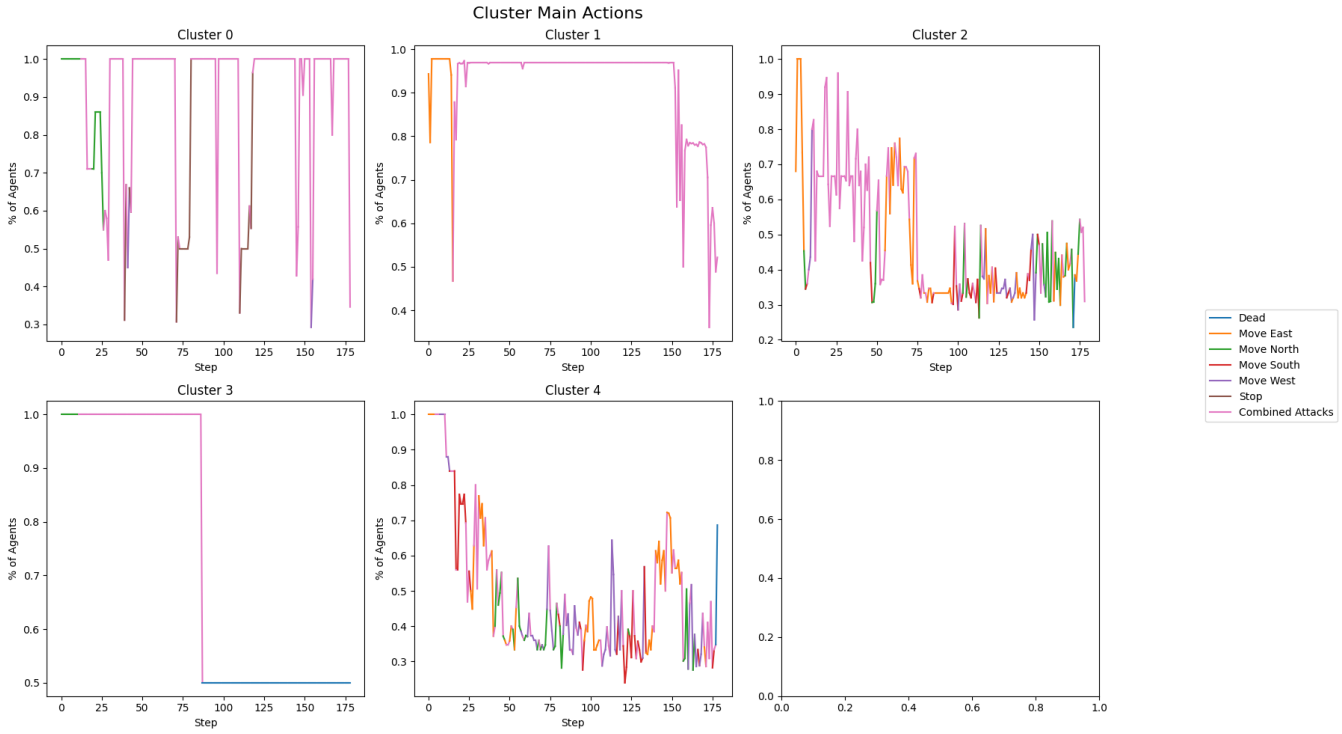

Figure 6: Main Action Graph for all cluster centers.

might be different, because the agents spread around in a different pace, or to different direction, but when looking at the whole picture, both represent the strategy of spreading around the enemy units.

Cluster 3 represents a very simple strategy. All agents in this cluster perform the same action in unity. According to table 2, only the heuristic of scenario 2s_vs_1sc was assigned to Cluster 3. This scenario contains a relatively hard fight, and all trajectories of heuristic runs on this scenario resulted in the agents losing. On the other hand, the heuristic of scenario 8m was assigned to the same cluster as QMIX's 8m, which is Cluster 1. Other than focusing fire and not moving, something that identifies Cluster 3 is the large percentage of dead agent, which helps differentiating it from other scenarios and does not directly teaches about the strategy itself. Therefore, we tried removing the "Dead" action from all trajectories, and indeed all heuristic trajectories were then assigned to Cluster 1, which represents a simpler strategy.

Other changes that were the result of removing the "Dead" Action were:

1. Easy and hard 2s_vs_1sc were assigned to Cluster 1 instead of Cluster 0.
2. Easy MMM and hard MMM were each assigned to a new cluster of their own, as it is the only scenario in which the "Healing" action is performed, and removing "Dead" action emphasizes that.

These changes, which result in a slight changes in the assignment of trajectories to clusters, show that most of our method is mostly robust and allows to recognize large-scaled strategies. Nonetheless, slight changes, such as adjusting some of the actions, normalizing the data in a different way, as well as slight difference in the timing and pacing of performing the strategy, will result in slight changes in the cluster centers. It is something that should be taken into account when using the method we suggested here.

4 Conclusion and Discussion

In this project, we managed to train multiple models to solve problem scenarios in the Starcraft II domain. Those solutions represented strategies learned by the models. We then compared the different strategies learned, and gained insight into the characteristics of a winning strategy in each of the scenarios. We also saw that some scenarios employ very similar strategies, while also seeing scenarios where pinning down a single strategy is not as simple.

Overall, the project shows that different scenarios require different strategies, and that strategies carry identifying and distinguishable characteristics which allow us to cluster them together even without knowing that they are from the same scenario.

5 Future Work

For future work, we propose two possible directions:

Strategies of Different Models In this work, only QMIX was utilized. This raises the question, "do different models learn different strategies?" Of course, if a model fails to win the level consistently, then its "learned" strategy is going to be very different than a model that does win. But, between models with similar win rate, do the agents behave similarly? Due

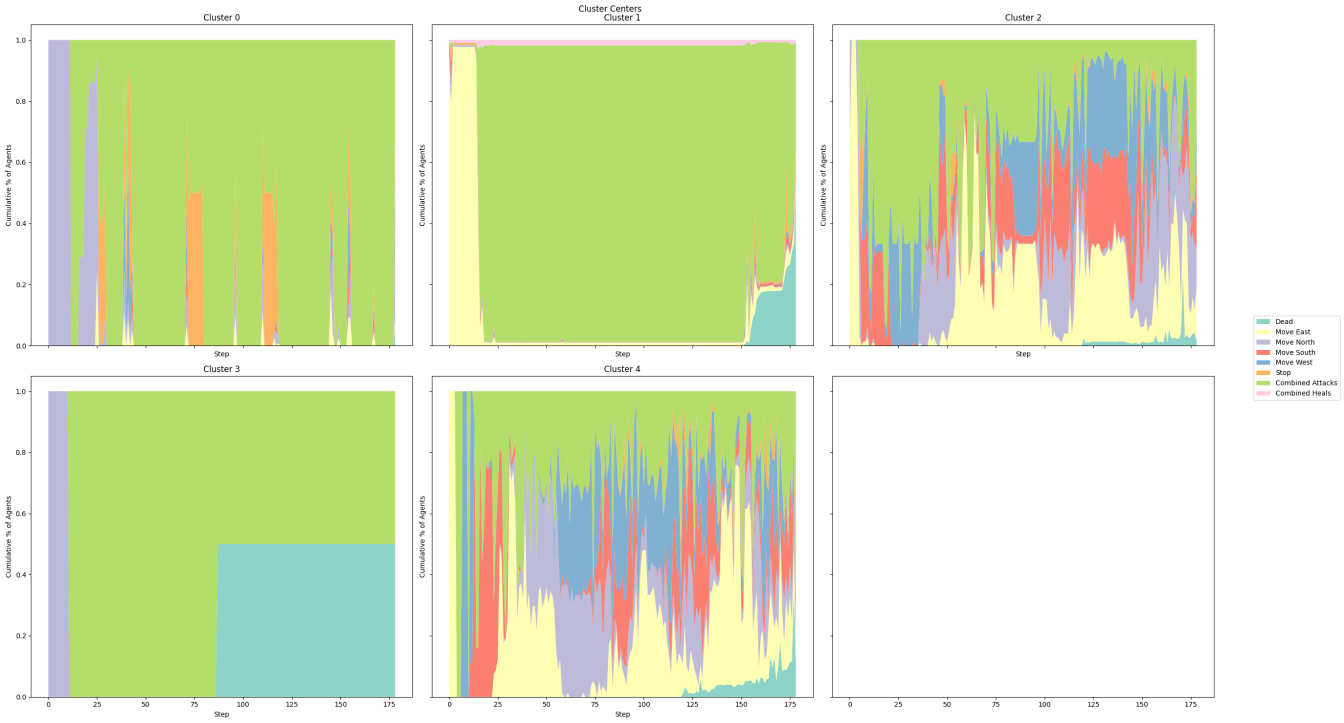

Figure 7: Stacked cluster centers.

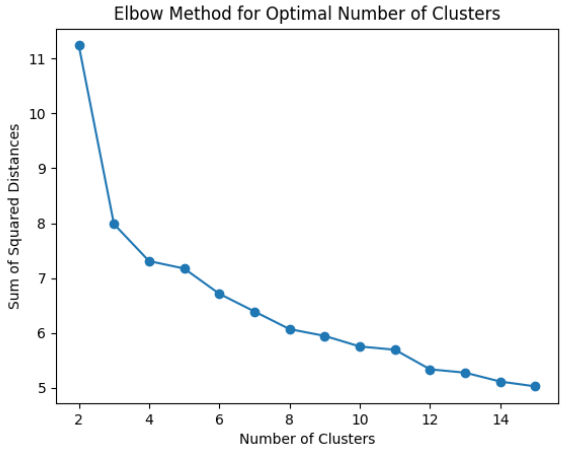

Figure 8: Elbow method for Optimal number of clusters

to the limitations of SMAC 1 in scenario variation, SMAC 2[4] [Ellis *et al.*, 2022] could be utilizied to generate more difficult and varied scenarios.

Transferable Learning between Similar Strategies While most scenarios require their own unique strategy, some similar scenarios were clustered together. This suggests a relatively similar strategy taken by the agents. Can we then take a trained model from one scenario and apply it on another one from the same cluster? This will require some adaptation of the model, as there might not be the same number of agents, or a slightly different map. How successful such transfer will be, and can we predict the success rate depending on features of the strategy itself?

Automatic Cluster Labeling In this project, we showed two representations that allow analyzing different clusters and strategies. Nonetheless, manually labeling the clusters according to the actions that were performed can lead to inaccuracies, and is very time consuming in case that there are many strategies or clusters. Combining our method with a language model that learns to analyze cluster centers in order to label and caption them could be helpful, and might reveal strategies that were missed before.

[4]https://github.com/oxwhirl/smacv2

References

[Ellis *et al.*, 2022] Benjamin Ellis, Skander Moalla, Mikayel Samvelyan, Mingfei Sun, Anuj Mahajan, Jakob N. Foerster, and Shimon Whiteson. Smacv2: An improved benchmark for cooperative multi-agent reinforcement learning, 2022.

[Hu *et al.*, 2021] Jian Hu, Siyang Jiang, Seth Austin Harding, Haibin Wu, and Shih wei Liao. Rethinking the implementation tricks and monotonicity constraint in cooperative multi-agent reinforcement learning. *arXiv:2102.03479*, 2021.

[Rashid *et al.*, 2018] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value

function factorisation for deep multi-agent reinforcement learning, 2018.

[Samvelyan *et al.*, 2019] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philiph H. S. Torr, Jakob Foerster, and Shimon Whiteson. The StarCraft Multi-Agent Challenge. *CoRR*, abs/1902.04043, 2019.