MARL-LNS: Multi-agent Deep Reinforcement Learning via Large Neighborhoods

Weizhe Chen, Sven Koenig, Bistra Dilkina
Thomas Lord Department of Computer Science
University of Southern California
Los Angeles, CA 90089
{weizhech, skoenig, dilkina}@usc.edu

Abstract

Cooperative multi-agent reinforcement learning (MARL) has been an increasingly important research topic in the last half-decade because of its great potential for real-world applications. Because of the curse of dimensionality, the popular "centralized training decentralized execution" framework requires a long time in training, yet still cannot converge efficiently. In this paper, we propose a general training framework, MARL-LNS, to address these issues by training on alternating subsets of agents using existing deep MARL algorithms as low-level trainers, while do not involve any additional parameters to be trained. Based on this framework, we provide two algorithms, random large neighborhood search (RLNS) and batch large neighborhood search (BLNS) which alternate the subsets of agents differently. We test our algorithms on both the StarCraft Multi-Agent Challenge and Google Research Football, showing that our algorithms can significantly reduce training time while reaching a better final skill level compared to the original low-level trainers in most cases.

1 Introduction

In recent years, multi-agent reinforcement learning (MARL) has split into cooperative multi-agent reinforcement learning and competitive multi-agent reinforcement learning. Competitive MARL has many theoretical guarantees following the previous studies in game theory, and has substantial success in domains like Poker [1], Diplomacy [2, 3]. On the other hand, cooperative multi-agent reinforcement learning focuses more on training a group of agents and making them coordinate with each other when everyone shares the same goal, or their goals need coordination. The cooperative MARL community has also succeeded in many real-world applications like autonomous driving [4], swarm control [5], autonomous warehouse [6, 7], and traffic scheduling [8].

While previous research has shown that MARL can converge to a good policy that wins in a game or finish some tasks in a given time, how to train the agent efficiently remains a very big challenge. Existing cooperative MARL algorithms face difficulties in the iterative process of simultaneous policy updates by all agents, especially in scenarios involving a large number of agents. This challenge arises from the curse of dimensionality, wherein the total size of joint state-action spaces exponentially grows with the number of agents involved. To address this issue, researchers have introduced agent priorities or orders to let some agents know more than just the joint state, such as knowing what actions other agents are going to choose, and what other agents are going to learn in the next phase [9]. In these cases, only one agent is supposed to choose an action during the learning based on the raw input, and other agents can rely on the additional information that the previous agents provide. However, this line of research usually requires a more centralized training scheme which in turn needs much more effort to make the policy be deployed to execution.

Preprint. Under review.

In this paper, we propose a new learning framework that reduces the time used in the training process of cooperative MARL without harming the performance of the final converged policy. We split the training into multiple iterations, which we called *LNS iterations*, and we only consider the training introduced by a fixed group of agents in each iteration. We choose a subset of agents that are used in training for a certain number of training iterations using existing deep MARL algorithms to update the neural networks, and then alternate the subsets of agents so that we do not overfit the policy to one certain subgroup of agents while we need to control all of them. We call the group of agents we are considering the *neighborhood*, and call our framework large neighborhood search (MARL-LNS), whose name comes from similar methods used in combinatorial optimization [10]. Since we only change the training in high-level, combining our MARL-LNS framework with any existing MARL algorithm is quite easy and simple. Our framework successfully reduces the dimension of the joint action space that we are considering at the same time from $|A|^n$, where n is the number of agents, to a joint action space of $|A|^m$, where m is the size of the neighborhood of agents that we consider in each iteration of the neighborhood. We provide a theoretical analysis that after multiple LNS iterations, the optimal action learned in this reduced joint action space could still hold the convergence guarantee provided by the low-level MARL algorithm.

With our framework, we provide two simple yet powerful algorithms, RLNS and BLNS, using MAPPO [11] as the low-level trainer. To show the capability of our framework, our algorithms rely on random choices of which group of agents is used in training and do not require any hand-crafted or learned heuristic to select the neighborhood. Thus, our algorithms also do not introduce any additional parameters to the training.

We test our algorithms in many scenarios in the popular StarCraft Multi-Agent Challenge (SMAC) [12] and Google Research Football (GRF) [13] environments, and show that our simple algorithms can reach at least the same level of performance while we can be substantially faster than the baselines in terms of wall clock time. We provide an ablation study on how the number of agents in the training neighborhood at each period affects the performance of the algorithms and the training time.

2 Related Works

Cooperative multi-agent reinforcement learning has been a popular research topic in the past few years. The most popular solution framework is the centralized training decentralized execution (CTDE) framework, in which two of the most popular lines of research are the value-based research like VDN [14], QMIX [15], QTRAN[16], WQMIX [17], QPLEX[18], and policy-based research like MADDPG [19], COMA[20] and MAPPO [11]. While many of these works are performing well in many scenarios, the curse of dimensionality problem, posed by the fact that the joint state-action space is exponentially big in the number of agents, is still a challenge to the convergence of MARL algorithms. Therefore, recently researchers have been trying to reduce the state space size by introducing orders to the agents. Agents can make decisions based on other agents' known actions [21, 22, 23, 24], or get trained in turns so that there is always one agent that is currently being trained [25]. Other researchers have directly taken the learning process of other agents into account, and proposed to change the update formulation to address them, resulting in algorithms like LOLA [26], POLA[27] and HLA[28]. Our algorithms train part of the total set of agents at a time, which is different than reducing the training to one agent at each time, and we are not actually letting the agents implicitly know that in the framework. A few recent works propose using only local neighborhood information at each timestep to keep the joint state-action space small [29]. Our algorithm framework is different from those by not changing the neighborhood used at each timestep in one episode, but only changing it after enough training iterations. In this way, we do not need to build a graph and consider distances in how the neighborhood is formed, which is hard in many tasks. Other works like VAST [30] proposed to factorize the coordination into a sum over a few smaller sub-groups with smaller joint-action space, while our algorithm will only train one sub-group at one time so this small joint-action space is very stable in each LNS iteration, while also using shorter training time in each training iteration. Recently, DER[31] also proposed to use part of the data in training, but they focused on training the policy efficiently in the beginning, while we are focusing on the overall efficiency and effectiveness.

Our large neighborhood search algorithm shares the same name as a group of similar algorithms used in combinatorial optimization (CO) and multi-agent path-finding (MAPF) communities. Large neighborhood search has been extensively used in combinatorial optimization problems since proposed [10], such as vehicle routing [32] and solving mixed integer linear programming [33, 34]. Similar ideas have also been used to help convergence in game theory by fixing a group of agents [35]. Recently, the same technique has been introduced to multi-agent path finding, where they fix the path of a group of agents and replan the path of other agents at each iteration [36, 37, 38]. In these algorithms, the decisions of the variables/agents in the chosen neighborhood are updated such that this "move" results in a better overall solution to the optimization problem. Even if the name is the same, our framework is completely different from theirs: these optimization domains do not have the trouble of learning, and they do not need to worry about the result of applying a 'move' in the chosen neighborhood (in our case, resulting in an updated policy) leading to a worse overall solution, which is one of the big problems the uncertainty of MARL can have; the existing papers mostly solve the subproblem implied by the neighborhood from scratch at each iteration, while our framework does not require the agents to be trained from scratch so that we do not need to resample all the trajectories and waste time on early near-random samples.

In one special case of our algorithm, when the size of the neighborhood is 1, our algorithm becomes iterative learning, which has been widely used in equilibrium findings, for example, Nash Equilibrium[39], Strong Stackelberg Equilibrium [40], Perfect Bayesian Nash Equilibrium[41], and distributed partially observable Markov decision process [35]. While their works mostly use iterations to make the training more stable and smooth because everyone has their own reward, our work also focuses on reducing the time in each training iteration.

3 Preliminaries

3.1 Dec-POMDP

In multi-agent reinforcement learning, agents interact with the environment, which is formulated as a decentralized partially observable Markov decision process (Dec-POMDP). Formally, a Dec-POMDP is defined by a 7-tuple $\langle S, A, O, R, P, n, \gamma \rangle$. S is the state space. $A = (A_1, A_2, \ldots, A_n)$ is the action space for each agent. $O(s, i)$ is the observation function that takes the current state and the agent number as inputs and outputs the specific observation o_i for agent i. $P(s'|s, A)$ is the transition probability from s to s' given the joint action $a = (a_1, a_2, \ldots, a_n)$ for all n agents. $R(s, a)$ denotes the shared reward function that every agent received. γ is the discount factor used in calculating the cumulative reward. In most MARL papers, an extra function $D(s, a)$ is introduced, which maps the state and action together to a binary signal that indicates whether the process is complete (done) after doing the action at the current state. During the episode, agents use a policy $\pi_i(a_i|o_i)$ to produce an action a_i from the local observation o_i. The environment starts with an initial state s^1, and stops when the environment provides a true in the *done* signal. The target of the learning is to find a policy within the valid policy space $\pi = (\pi_1, \pi_2, \ldots, \pi_n) \in \Pi$ that optimizes the discounted accumulated reward $\mathcal{J} = \mathbb{E}_{s^t, a^t \sim \pi} \sum_t \gamma^t \cdot r^t$, where s^t is the state at timestep t, $r^t = R(s^t, a^t)$, and $a^t = (a_1^t, a_2^t, \ldots, a_n^t)$ is the joint action at timestep t.

3.2 Centralized Training Decentralized Execution

One of the most popular training frameworks for MARL in recent years is the Centralized Training Decentralized Execution (CTDE) framework. The framework can be further split into two branches, namely, value-based algorithms like QMIX [15], and policy-based algorithms like MAPPO [11]. In this paper, we mainly focus on the second one, the policy-based algorithms.

In these algorithms, a trajectory τ is a list of elements that contains all the information of what happens from the start to the end of one run of the environment, typically $\tau =< s^1, a^1, r^1, s^2, a^2, r^2, \ldots, s^t, a^t, r^t >$. During the training of CTDE algorithms, the trajectories are split into trajectories in the view of every agent $\tau = (\tau_1, \tau_2, \ldots, \tau_n)$, which contains useful information in terms of each agent and forms an individual trajectory. The information included in each τ_i depends on the algorithm that is used, for example, MADDPG needs to train a centralized value function, τ_i is the same as τ. However, for MAPPO with GAE, the value function only needs the centralized state but not the joint action, so $\tau_i =< s^1, a_i^1, r^1, s^2, a_i^2, r^2, \ldots, s^t, a_i^t, r^t >$ which reduce the joint action as needed.

Algorithm 1 Large Neighborhood Search for MARL (MARL-LNS)

```
1:  Initialize value network $V$ and policy network $\pi$.
2:  repeat
3:      Choose the neighborhood $R = NeighborhoodSelect()$.
4:      repeat
5:          Reset the replay buffer.
6:          repeat
7:              Sample trajectory $\tau = (\tau_1, \tau_2, \tau_3, \ldots, \tau_n)$ from the environment according to $\pi$.
8:              Save $\tau_{r_1}, \tau_{r_2}, \ldots, \tau_{r_m}$ to the replay buffer.
9:          until Sampled Enough Trajectories.
10:         Train $V$ and $\pi$ using the replay buffer with low-level algorithm.
11:     until Trained Enough Iterations.
12: until Reach The Stopping Criteria for LNS
```

4 Large Neighborhood Search for MARL

4.1 Large Neighborhood Search Framework

Large Neighborhood search is a popular meta-heuristic used in combinatorial optimization and multi-agent path finding for finding good solutions in challenging problems where finding the optimal solution is extremely time-consuming and infeasible in reality. Starting from an initial solution, part of the solution is selected, called a *neighborhood*, and then destroyed. The optimizer then needs to rebuild the solution for the parts in the neighborhood while knowing the solution for the remaining parts and freezing their values for efficiency. Because a solution needs to also be feasible with respect to hard constraints in the optimization domains, LNS mostly serves as a good method to enable the possibility of starting with a fast but far-from-optimal algorithm for finding a feasible solution and then improving the solution using other algorithms that is slow but has a better solution quality until the time limit. This normally results in an anytime algorithm, which is favorable for real-world concerns. However, in MARL, the policy at any training time can be used, making the training process naturally an anytime algorithm, and the purpose of LNS in our paper is for a shorter training time and better solution quality.

While the high-level idea can lead to many completely different algorithms, we now give a detailed description of our framework in MARL. During training, we always keep a group of m agents, where m is a hyperparameter that we can choose. We call this group of agent a *neighborhood* $R = \{r_1, r_2, \ldots, r_m\}$, and m is the neighborhood size. Then we first sample a batch of data from the environment. During the sampling process, we decouple full information trajectory τ to n subtrajectories $(\tau_1, \tau_2, \ldots, \tau_n)$, where τ_i is to be used to train agent i respectively. Then we only keep the trajectories for the agents that are in the neighborhood, and put them into the replay buffer that is later used for training. Thus, the trajectories that we save are the set of $\{\tau_{r_1}, \tau_{r_2}, \ldots, \tau_{r_m}\}$. We do one training when enough data for one batch is sampled, using existing training algorithms like MAPPO. In this way, the training data only contains the information used to train the agents in the neighborhood. We repeat the trajectory sampling process several times until a new group of agents is resampled as the new neighborhood. We call the sampling of trajectories and training for one fixed neighborhood an *LNS iteration*. The pseudo-code of our algorithm is provided in Alg. 1.

Unlike many existing works focusing on data efficiency, our algorithms gain efficiency by training faster in each batch because fewer data are now used for backpropagation. Also, when every agent has the same action space, the total joint action space we have is reduced from $|A|^n$ to $|A|^m$. This means that in each LNS iteration, the algorithm needs a shorter time to explore the possible joint actions before it converges to a stationary point.

As a high-level approach, our algorithm does not have the specification on how the data is used for training, and how coordination is solved in the framework. Because of this, the lower-level MARL algorithm, which uses the data to train the neural network, is very flexible. In this paper, we choose to use MAPPO [11] as the low-level algorithm as an example. In MAPPO, there is a centralized value function that concatenates all observations from all agents in the environment. Even if some agents are not included in the neighborhood and thus not used in training, their observations are still kept in the input of the value function as global state information, so we do not need to retrain from scratch the value function in each LNS iteration. We want to clarify that although in this paper we only consider a policy-based algorithm as low-level algorithm, value-based frameworks like QMIX [15] can also use our framework by only using the trajectory of agents in the neighborhood to train the local Q networks and fix the mixing network until the end of each iteration.

Next, we give theoretically analyze our algorithm to show that the convergence guarantee will not be affected by the introduced framework. Our algorithm can be reduced to a block gradient descent algorithm (BCD) by letting each variable used in the optimization is the policy of each agent, and the objective value is our reward function. BCD is studied a lot by optimization theory researchers [], and its convergence rate is proved under different conditions. Here we provide a convergence guarantee that fits the multi-agent deep reinforcement learning framework the most:

Theorem 4.1. *[42] Assume the expected cumulative reward function $\mathcal{J}$ is continuously differentiable with Lipschitz gradient and convex in each neighborhood partition, and the training by the low-level algorithm guarantees that the training happening on the i-th neighborhood is bounded by a high-dimension vector w_i on the joint policy space Π. Define the optimality gap as $\Delta_i(\pi) := sup_{\hat{\pi}\in\Pi, |\hat{\pi}-\pi|\leq c' w_i} J_{\hat{\pi}} - J_{\pi}$, where c' is a constant. Suppose $\sum_{i=1}^{\infty} w_i^2 < \infty$, and let the policy after the k-th LNS iteration be π^k. Then the following hold:*

1. *If $\sum_{i=1}^{\infty} |w_i| = \infty$, and $\Delta_i = O(|\pi - \hat{\pi}|^2)$, then the training can converge to a stationary point.*

2. *If the optimality gap is uniformly summable, i.e., $\sum_{i=1}^{\infty} \Delta_i < \infty$, then there exists some constant $c > 0$ such that for $i \geq 1$,*

$$min_{1\leq k\leq i} \sup_{\pi_0\in\Pi} [-\inf_{\pi\in\Pi} \langle \nabla \mathcal{J}_{\pi^k}, \frac{\pi - \pi^k}{|\pi - \pi^k|} \rangle] \leq \frac{c}{\sum_{k=1}^{i} w_k} \tag{1}$$

Specifically, the assumptions on w_i are common assumptions for convergence in MARL, and are normally handled by the learning rate decay mechanism in the learning optimizer together with the clip mechanism in reinforcement learning algorithms like TRPO and PPO. Furthermore, because the expected reward function $\mathcal{J}$ is based on policy rather than action, the continuously differentiable condition is also satisfied in environments with a continuous policy space. This theorem guarantees that the convergence of MARL-LNS is irrelevant to the neighborhood size m, thus the final policy learned by MARL-LNS is the same as the vanilla policy optimization, which is the special case of the size of the neighborhood being the same as the number of agents. However, in reality, the learned policy may be worse than the policy learned by the low-level algorithm if the learning rate and exploration are not handled properly.

Furthermore, because our framework does not control the training part, which calculates how the data is turned into loss to train the network, our algorithm holds any monotonic improvement bound that is provided by the low-level algorithm.

4.2 Random Large Neighborhood Search

While many previous works of large neighborhood search in CO and MAPF focus a lot on neighborhood selection, in this paper we show the capability of our framework by choosing the neighborhood randomly and do not introduce any hand-crafted heuristics. Such a random-based algorithm has been shown to work decently in combinatorial optimization [43, 34].

For simplicity, we assume that the neighborhood size m is fixed throughout the training, and the total training episodes are split uniformly for different neighborhoods, so the criteria of having trained enough iterations for the neighborhood in Alg. 1 is now replaced by comparing the number of training that has been done to the number of training in one neighborhood. The neighborhood selection part is instantiated with uniformly sample m agents from 1,..,n without replacement as $random.choice$ do in NumPy . Given a large number of training iterations in training, the random play a similar role to stochastic gradient descent which also randomly samples part of the training data for training. Typically, a random-based algorithm will converge with many fluctuations, but fluctuations can also serve as a good way of exploration in multi-agent reinforcement learning, and it is not a problem for converging. We provide the pseudo-code of Random Large Neighborhood Search (RLNS) in Alg. 2.

Algorithm 2 Random Large Neighborhood Search (RLNS)

```
1: Input: neighborhood size $m$
2: Initialize value network $V$ and policy network $\pi$.
3: repeat
4:     $R = random.choice(n, m)$.
5:     repeat
6:         Reset the replay buffer.
7:         repeat
8:             Sample trajectories $\tau = (\tau_1, \tau_2, \tau_3, \dots, \tau_n)$ from the environment according to $\pi$.
9:             Save $\tau_{r_1}, \tau_{r_2}, \dots, \tau_{r_m}$ to the replay buffer.
10:        until Sampled $Buffer\_length$ trajectories.
11:        Train $V$ and $\pi$ with the replay buffer using MAPPO.
12:    until Trained $N_{Training\_per\_neighborhood}$ rounds.
13: until Has done $N_{LNS\_iterations}$ Iterations.
```

Algorithm 3 Batch Large Neighborhood Search (BLNS)

```
1: Input: neighborhood size $m$.
2: Initialize value network $V$ and policy network $\pi$.
3: Initialize a permutation of n: $(p_1, p_2, \dots, p_n)$, and a neighborhood index $idx_p = 1$.
4: repeat
5:     $R = \{\}$.
6:     while $|R| < m$ do
7:         $R = R \cup p_{idx_p}$.
8:         $idx_p = idx_p$ mod $n + 1$
9:     end while
10:    repeat
11:        Do one round of training. (The same code as Line 6-11 in Algo. 2.)
12:    until Trained $N_{Training\_per\_neighborhood}$ rounds.
13: until Has done $N_{LNS\_iterations}$ Iterations.
```

4.3 Batch Large Neighborhood Search

While pure random can introduce a lot of variance to the training, here we also provide a batch-based large neighborhood search (BLNS) algorithm. Unlike RLNS, we create one permutation $(p_1, p_2, \dots, p_n)$ of all agents before any training starts. Again for simplicity, the permutation is created randomly in this paper. After creating the permutation, we select the agents in order whenever we want to select the next group of neighborhoods. In other word, given the fixed neighborhood size m, the first neighborhood would be $\{p_1, p_2, \dots, p_m\}$, the second would be $\{p_{m+1}, p_{m+2}, \dots, p_{2m}\}$, and keep going like this. If m cannot be divided by n, the neighborhood that includes the last agent p_n will also include agent p_1, and keep going from p_2, p_3 to p_n again. All other parts of BLNS are the same as RLNS. The pseudo-code is provided in Alg. 3.

BLNS, as its name goes, is very similar to batch gradient descent. With a batch-like training framework, the training is smoother than the pure random-based algorithm RLNS and can provide a better final policy in most cases. Our algorithm differs from the batch gradient descent in that our training samples are not independent and identically distributed (i.i.d.) over the possible space.

5 Experiments

5.1 Experimental Settings

In this paper, we test our results on both StarCraft Multi-Agent Challenge (SMAC) and Google Research Football (GRF) environments which are shown in Fig. 1 and Fig. 2. While we will provide the settings specific for each environment later, we first provide some common information here.

We have used some of the most common settings used in cooperative MARL. We use parameter sharing between agents because it has been shown to improve the training efficiency while not harming

```
CHEAT Local Player: KillUnitByTag 3670030
CHEAT Local Player: KillUnitByTag 3407896
CHEAT Local Player: KillUnitByTag 1310734
CHEAT Local Player: KillUnitByTag 1572894
```

Figure 1: An illustration of the 3s5z vs 3s6z map in SMAC. On the left-hand side are the units that we need to control, which include 3 Stalkers and 5 Zealots.

Figure 2: The corner scenario in GRF environment, which is a standard corner-kick scenario except for the agent on the corner can carry the ball himself.

Table 1: Median evaluation win rate and standard deviation on SMAC testbed. We use tables provided in [11] and [46] for baseline numbers for MAPPO, IPPO, Qmix, and CoPPO.

	MAPPO	IPPO	QMix	CoPPO	RLNS	BLNS
5mvs6m	89.1 (2.5)	87.5 (2.3)	75.8 (3.7)	84.4 (2.1)	**96.9 (8.2)**	**96.9 (3.6)**
MMM2	90.6 (2.8)	86.7 (7.3)	87.5 (2.6)	90.6 (6.9)	**96.9 (31.8)**	**96.9 (4.7)**
3s5zvs3s6z	84.4 (34.0)	82.8 (19.1)	82.8 (5.3)	84.4 (2.9)	**87.5 (44.0)**	12.6 (31.8)
27mvs30m	**93.8 (2.4)**	69.5 (11.8)	39.1 (9.8)	**93.8 (2.2)**	90.6 (2.5)	**93.8 (7.2)**
10mvs11m	**96.9 (4.8)**	93.0 (7.4)	95.3 (1.0)	**96.9 (2.6)**	93.8 (5.3)	**96.9 (2.4)**

the performance [44]. We use some common practice tricks in MAPPO, including Generalized Advantage Estimation (GAE) [45] with advantage normalization, value clipping, and including both local-agent specific features and global features in the value function [11]. We use the same hyperparameters and network designs as our low-level training algorithm MAPPO [11], and the full table of hyperparameters is provided in the appendix. For our algorithms, instead of setting a number of how many times of training is used for each LNS iteration, we use an equivalent version of providing the total number of different neighbors, so the number of how many times of training are just the total episodes divided by the number of different neighborhoods, i.e., $N_{Training\_per\_neighborhood} = N_{LNS\_iterations}/N_T$, and N_T is the new hyperparameter we control. By default, the neighborhood size is half of the total number of agents. All results are trained on servers with a 16-cores xeon-6130 2.10 GHz CPU with 64GB memory, and an NVIDIA V100 24GB GPU.

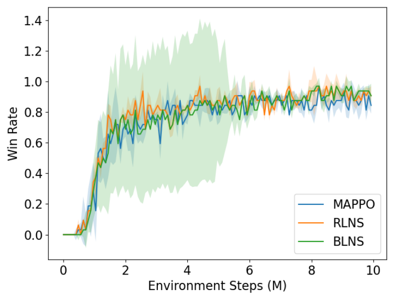
5.2 SMAC Testbed

We test our algorithms on 5 different random seeds. For each random seed, we evaluate our results following previous works: we compute the win rate over 32 evaluation games after each training iteration and take the median of the final ten evaluation win rates as the performance. We compare our algorithm with CoPPO, MAPPO, IPPO and QMIX. We only test our results in scenarios that are classified as hard or super-hard, because many easy scenarios have been perfectly solved, and our algorithm will become iterative training, which has been studied a lot, in scenarios that include only 2 or 3 agents.

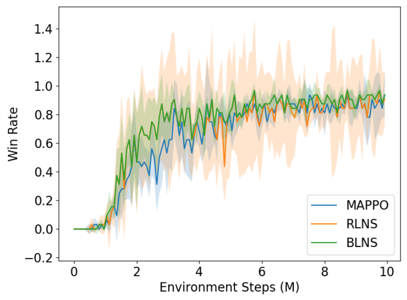
We report our results in Table. 1 and Fig. 3. We observe that the performance of our algorithms is at least at the same level as the current algorithms in terms of win rate, while actually getting a higher final win rate in difficult scenarios like 5mvs6m and MMM2. Furthermore, RLNS generally have a bigger variance than BLNS, which is coming from the pure-random-based neighborhood selection, but this random also enables RLNS to get good policy in the very hard 3s5zvs3s6z scenario where BLNS fails in 3 seeds and ends up with a low median value.

5.2.1 Ablation Study on Neighborhood Size

After showing that our algorithms are good in terms of learning the policies, we now use an ablation study on neighborhood size to show how our method provides flexibility to trade off a tiny win rate for faster training time. As a special case, when changing the neighborhood size to 1, BLNS is iterative training, and when the neighborhood size is as big as the total number of agents, our framework is



(a) 5M vs 6M



(b) MMM2

Figure 3: Training curves for RLNS and BLNS compared to MAPPO on some SMAC scenarios.

Table 2: Median value and the standard deviation on evaluation win rate for RLNS with different neighborhood sizes m, together with their average training time and average updating time (training time include both sampling time and updating time) for 1k episodes on 27m_vs_30m scenario on the SMAC testbed. Specifically, when $m = 27$, RLNS will automatically downgrade to MAPPO.

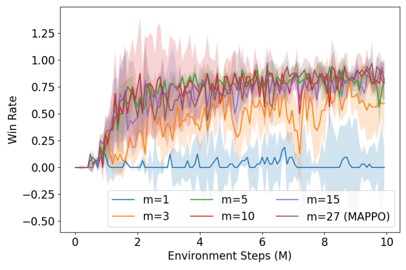
m	1	3	5	10	15	27 (MAPPO)
Win Rate (%)	3.1 (3.9)	62.5 (28.3)	87.5 (5.3)	90.6 (8.2)	93.8 (7.2)	93.8 (3.8)
Training Time (s)	7.14	7.35	7.37	8.22	9.29	9.52
Updating Time (s)	2.10	2.31	2.35	4.02	5.10	5.33

the same as the low-level algorithm MAPPO. We did not change the total environment steps used in training because we do not see any benefit in doing that.

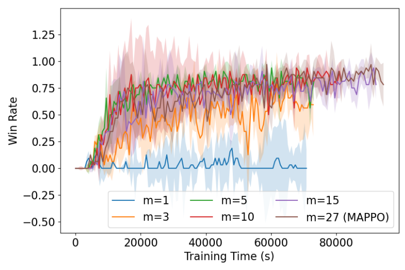
We test BLNS on the 27mvs30m scenario, because it has the biggest number of agents. We show our results in Table. 2 and Fig. 4. We observe that when increasing the neighborhood size, the final win rate is improving, while the time usage is also bigger. When we set the neighborhood size m to 10, the final performance is within one standard deviation of the low-level algorithm, while the training is 15% faster. And if we set the neighborhood size m to 5, the final performance is within two standard deviations of MAPPO, with only 77% of the original training time used. We also observe from Fig. 4b that given a limited training time that is not enough for convergence, by reducing the neighborhood size to 5 or 10, the learning can be significantly more efficient than using more. This saving in time is because even though our CPUs are more old-aged compared to our GPUs, the sampling time in CPUs is only taking 44% of the total training time used to train MAPPO as shown in Table. 2. All other time is spent on transferring data between CPU and GPU and updating the neural networks on GPU, which are the time related to updating that can be largely saved by removing part of the training data.

Table 3: The average evaluation win rate of BLNS in different settings compared to RLNS and other baselines on GRF. We have let $N_{Training\_per\_neighborhood} = N_{LNS\_iterations}/N_T$. We use the numbers in the MAPPO paper [11] for baselines under the line.

	Win Rate (%)
BLNS ($m = 7, N_T = 20$)	65.6(8.1)
BLNS($m = 5, N_T = 20$)	57.4(6.0)
BLNS($m = 7, N_T = 10$)	42.4(3.1)
BLNS($m = 7, N_T = 5$)	42.2(1.8)
RLNS($m = 7, N_T = 20$)	58.0(13.5)
MAPPO	65.53(2.19)
QMix	16.10(3.00)
CDS	3.80(0.54)



(a) Win rate corresponds to environment step.



(b) Win rate corresponds to wall clock time.

Figure 4: The training curve of BLNS on SMAC when different neighborhood size m is given.

5.3 GRF Testbed

We test our algorithms on 5 different random seeds. We evaluate our results in a similar way as we do in SMAC, but we change as common practice: instead of evaluating in 32 evaluation games, we evaluate the result in 100 rollouts, and instead of reporting the median value we report the mean value. Because most scenarios in this testbed have a number of agents less than 6, we only test the algorithm on the corner scenario, which is shown in Fig. 2.

While testing our algorithm on this testbed, we found that the number of different neighborhoods used in our algorithm is also quite important for efficient training. While fixing the number of total episodes used in the experiment, changing the number of iterations $N_{LNS\_iterations}$ to a large number will reduce the number of training in each iteration, and make none of the neighborhoods actually get close to the equilibrium in it. On the other hand, setting $N_{LNS\_iterations}$ will fail to learn a policy that takes all agents into account. Thus, we need to find a good number in the middle.

The results are shown in Table. 3. Even if this test case is naturally heterogeneous, we observe that giving a good hyperparameter to BLNS will give our algorithm the same level of performance as MAPPO that is within one standard deviation, and applying the same group of hyperparameters to RLNS can learn the policy with larger variance. In this hyperparameters setting, BLNS and RLNS are trained 13% faster than MAPPO in time. We also find that the performance is optimal when the number of LNS iterations is 20, which is still not too big to make the neighborhood always change.

Comparing the effect on different hyperparameters feed into our models, the neighborhood size does make a difference as we have already shown in the SMAC testbed, but it is less important than the number of iterations in this testbed as long as m is not too small in GRF environment. However, such a change in the result caused by changing N_T is not observed in the SMAC testbed. We believe the number of iterations is closely related to the actual task, and we encourage future work to systematically study how this hyperparameter should be set.

6 Limitations and Potential Negative Impact

While our algorithms are performing well in SMAC and GRF, there are two major limitations in our algorithms. First, our algorithm framework MAPF-LNS relies on the fact that in big and complex environment, not all agents are used to fulfill a single task, but agents are learned to split the job into different tasks and do their own parts. Because of this, the Multi-domain Gaussian Squeeze (MGS) environment used in the Qtran paper [16] is one counter-example that requires strong cooperation within all agents and thus leads to unstable and slow convergence using our algorithms. Second, our current random-based neighborhood selection algorithms assume that all agents are similar, and no agents have a higher priority, higher importance, or significant difference from others in the environment. This is not applicable to some real-world scenarios where some agents should have higher priority, for example, a fire engine in the traffic system. For the same reason, the marginal distribution could be ignored during training which may lead to fairness issues if the neighborhood selection is not taking these into account as current RLNS and BLNS do, and this could lead to a potential negative social impact. We do not see any other potential negative impact for MARL-LNS.

7 Conclusion

In this paper, we propose a novel large neighborhood search framework (MARL-LNS) for cooperative MARL. Building upon MARL-LNS, we design two algorithms RLNS and BLNS that select the neighborhood differently. We theoretically and empirically show that our algorithms can greatly improve training efficiency while not harming the final policy, and even help learn a better policy in hard scenarios like MMM2 in SMAC.

A Implementation Details for Experiments

Here we provide the hyperparameter table for our experiments. While most results in the experiment section are from previous paper [11] and [46], we only provide the hyperparameter for our algorithms.

Table. 4 provide the neighborhood size used specifically for each scenario, while Table. 5 provide other hyperparameters that exist in the low-level training algorithm MAPPO. Hyperparameters are not specified by search but by directly using the recommended variables used by the low-level training algorithm MAPPO, except for the number of parallel environments which is limited by the core of our server. In SMAC environments, the number of neighborhood iterations is set to 8.

	Neighborhood size m
5mvs6m	3
mmm2	2
3s5zvs3z6z	5
27mvs30m	15
10mvs11m	5

Table 4: Hyperparameter Table for neighborhood size used in RLNS and BLNS.

Hyperparameters	value
recurrent data chunk length	10
gradient clip norm	10.0
gae lamda	0.95
gamma	0.99
value loss	huber loss
huber delta	10.0
batch size num	envs × buffer length × num agents
mini batch size	batch size / mini-batch
optimizer	Adam
optimizer epsilon	1e-5
weight decay	0
network initialization	Orthogonal
use reward normalization	True
use feature normalization	True
num envs (SMAC)	8
num envs (GRF)	15
buffer length	400
num GRU layers	1
RNN hidden state dim	64
fc layer dim	64
num fc	2
num fc after	1

Table 5: Hyperparameter table for the MAPPO training part used in BLNS and RLNS.

B Additional Theoretical Guarantee on Convergence of MARL-LNS

In the main paper, we have provided the theorem that guarantees the convergence of MARL-LNS to be the same as the low-level algorithm. Here we provide a stronger theorem in the case that in each LNS iterations, the policy is updated to local optimal.

Theorem B.1. *[42] Assume the expected cumulative reward function $\mathcal{J}$ is continuously differentiable with Lipschitz gradient and convex in each neighborhood partition, and the training by the low-level algorithm guarantees that the training happening on the i-th neighborhood is bounded by some high-dimension vector w_i. Suppose $\sum_{i=1}^{\infty} w_i^2 < \infty$, then the following hold:*

1. *If $\sum_{i=1}^{\infty} |w_i| = \infty$, then for any initial starting point of training, the training can converge to a stationary point.*

2. *If π^k is optimized to optimal in each LNS iteration, then there exist some constant $c > 0$ such that for $i \geq 1$,*

$$min_{1 \leq k \leq i} \sup_{\pi_0 \in \Pi} [- \inf_{\pi \in \Pi} \langle \nabla \mathcal{J}_{\pi^k}, \frac{\pi - \pi^k}{|\pi - \pi^k|} \rangle] \leq \frac{c}{\sum_{k=1}^{i} w_k}$$

Compared to the one used in the main paper, this theorem is relaxing on the assumption of the optimality gap to be summable. However in general practice, always making the policy to optimized to optimal in each LNS iteration will lead to an extremely long training time, thus is less preferable.

References

[1] Noam Brown and Tuomas Sandholm. Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.

[2] Jonathan Gray, Adam Lerer, Anton Bakhtin, and Noam Brown. Human-level performance in no-press diplomacy via equilibrium search. In *International Conference on Learning Representations*, 2020.

[3] Meta Fundamental AI Research Diplomacy Team (FAIR)†, Anton Bakhtin, Noam Brown, Emily Dinan, Gabriele Farina, Colin Flaherty, Daniel Fried, Andrew Goff, Jonathan Gray, Hengyuan Hu, et al. Human-level play in the game of diplomacy by combining language models with strategic reasoning. *Science*, 378(6624):1067–1074, 2022.

[4] Ming Zhou, Jun Luo, Julian Villella, Yaodong Yang, David Rusu, Jiayu Miao, Weinan Zhang, Montgomery Alban, Iman Fadakar, Zheng Chen, et al. Smarts: Scalable multi-agent reinforcement learning training school for autonomous driving. *arXiv preprint arXiv:2010.09776*, 2020.

[5] Maximilian Hüttenrauch, Adrian Šošić, and Gerhard Neumann. Guided deep reinforcement learning for swarm systems. *arXiv preprint arXiv:1709.06011*, 2017.

[6] Guillaume Sartoretti, Justin Kerr, Yunfei Shi, Glenn Wagner, TK Satish Kumar, Sven Koenig, and Howie Choset. Primal: Pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robotics and Automation Letters*, 4(3):2378–2385, 2019.

[7] Mehul Damani, Zhiyao Luo, Emerson Wenzel, and Guillaume Sartoretti. Primal _2: Pathfinding via reinforcement and imitation multi-agent learning-lifelong. *IEEE Robotics and Automation Letters*, 6(2):2666–2673, 2021.

[8] Florian Laurent, Manuel Schneider, Christian Scheller, Jeremy Watson, Jiaoyang Li, Zhe Chen, Yi Zheng, Shao-Hung Chan, Konstantin Makhnev, Oleg Svidchenko, et al. Flatland competition 2020: Mapf and marl for efficient train coordination on a grid world. In *NeurIPS 2020 Competition and Demonstration Track*, pages 275–301. PMLR, 2021.

[9] Jakub Grudzien Kuba, Ruiqing Chen, Muning Wen, Ying Wen, Fanglei Sun, Jun Wang, and Yaodong Yang. Trust region policy optimisation in multi-agent reinforcement learning. *arXiv preprint arXiv:2109.11251*, 2021.

[10] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming—CP98: 4th International Conference, CP98 Pisa, Italy, October 26–30, 1998 Proceedings 4*, pages 417–431. Springer, 1998.

[11] Chao Yu, Akash Velu, Eugene Vinitsky, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative, multi-agent games. *arXiv preprint arXiv:2103.01955*, 2021.

[12] 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.

[13] Karol Kurach, Anton Raichuk, Piotr Stańczyk, Michał Zając, Olivier Bachem, Lasse Espeholt, Carlos Riquelme, Damien Vincent, Marcin Michalski, Olivier Bousquet, et al. Google research football: A novel reinforcement learning environment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 4501–4510, 2020.

[14] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

[15] Tabish Rashid, Mikayel Samvelyan, Christian Schröder de Witt, Gregory Farquhar, Jakob N. Foerster, and Shimon Whiteson. QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4292–4301. PMLR, 2018.

[16] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International conference on machine learning*, pages 5887–5896. PMLR, 2019.

[17] Tabish Rashid, Gregory Farquhar, Bei Peng, and Shimon Whiteson. Weighted qmix: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning. *Advances in neural information processing systems*, 33:10199–10210, 2020.

[18] Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. Qplex: Duplex dueling multi-agent q-learning. *arXiv preprint arXiv:2008.01062*, 2020.

[19] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

[20] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[21] Dimitri Bertsekas. Multiagent rollout algorithms and reinforcement learning. *arXiv preprint arXiv:1910.00120*, 2019.

[22] Jakub Grudzien Kuba, Muning Wen, Linghui Meng, Haifeng Zhang, David Mguni, Jun Wang, Yaodong Yang, et al. Settling the variance of multi-agent policy gradients. *Advances in Neural Information Processing Systems*, 34:13458–13470, 2021.

[23] Haifeng Zhang, Weizhe Chen, Zeren Huang, Minne Li, Yaodong Yang, Weinan Zhang, and Jun Wang. Bi-level actor-critic for multi-agent coordination. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7325–7332, 2020.

[24] Chuming Li, Jie Liu, Yinmin Zhang, Yuhong Wei, Yazhe Niu, Yaodong Yang, Yu Liu, and Wanli Ouyang. Ace: Cooperative multi-agent q-learning with bidirectional action-dependency. *arXiv preprint arXiv:2211.16068*, 2022.

[25] Jakub Grudzien Kuba, Xidong Feng, Shiyao Ding, Hao Dong, Jun Wang, and Yaodong Yang. Heterogeneous-agent mirror learning: A continuum of solutions to cooperative marl. *arXiv preprint arXiv:2208.01682*, 2022.

[26] Jakob N Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. *arXiv preprint arXiv:1709.04326*, 2017.

[27] Stephen Zhao, Chris Lu, Roger Baker Grosse, and Jakob Nicolaus Foerster. Proximal learning with opponent-learning awareness. *arXiv preprint arXiv:2210.10125*, 2022.

[28] Ariyan Bighashdel, Daan de Geus, Pavol Jancura, and Gijs Dubbelman. Coordinating fully-cooperative agents using hierarchical learning anticipation. *CoRR*, abs/2303.08307, 2023.

[29] Chengwei Zhang, Yu Tian, Zhibin Zhang, Wanli Xue, Xiaofei Xie, Tianpei Yang, Xin Ge, and Rong Chen. Neighborhood cooperative multiagent reinforcement learning for adaptive traffic signal control in epidemic regions. *IEEE Transactions on Intelligent Transportation Systems*, 23(12):25157–25168, 2022.

[30] Thomy Phan, Fabian Ritz, Lenz Belzner, Philipp Altmann, Thomas Gabor, and Claudia Linnhoff-Popien. Vast: Value function factorization with variable agent sub-teams. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pages 24018–24032. Curran Associates, Inc., 2021.

[31] Xunhan Hu, Jian Zhao, Wengang Zhou, and Houqiang Li. Discriminative experience replay for efficient multi-agent reinforcement learning. *CoRR*, abs/2301.10574, 2023.

[32] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4):455–472, 2006.

[33] Lluís-Miquel Munguía, Shabbir Ahmed, David A Bader, George L Nemhauser, and Yufen Shao. Alternating criteria search: a parallel large neighborhood search algorithm for mixed integer programs. *Computational Optimization and Applications*, 69(1):1–24, 2018.

[34] Jialin Song, Ravi Lanka, Yisong Yue, and Bistra Dilkina. A general large neighborhood search framework for solving integer programs. *arXiv*, 2020.

[35] Ranjit Nair, Milind Tambe, Makoto Yokoo, David V. Pynadath, and Stacy Marsella. Taming decentralized pomdps: Towards efficient policy computation for multiagent settings. In Georg Gottlob and Toby Walsh, editors, *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 705–711. Morgan Kaufmann, 2003.

[36] Jiaoyang Li, Zhe Chen, Daniel Harabor, P Stuckey, and Sven Koenig. Anytime multi-agent path finding via large neighborhood search. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.

[37] Taoan Huang, Jiaoyang Li, Sven Koenig, and Bistra Dilkina. Anytime multi-agent path finding via machine learning-guided large neighborhood search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 9368–9376, 2022.

[38] Jiaoyang Li, Zhe Chen, Daniel Harabor, Peter J Stuckey, and Sven Koenig. Mapf-lns2: fast repairing for multi-agent path finding via large neighborhood search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10256–10265, 2022.

[39] Constantinos Daskalakis, Rafael M Frongillo, Christos H Papadimitriou, George Pierrakos, and Gregory Valiant. On learning algorithms for nash equilibria. In *SAGT*, pages 114–125. Springer, 2010.

[40] Kai Wang, Qingyu Guo, Phebe Vayanos, Milind Tambe, and Bo An. Equilibrium refinement in security games with arbitrary scheduling constraints. In Elisabeth André, Sven Koenig, Mehdi Dastani, and Gita Sukthankar, editors, *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, pages 919–927. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, 2018.

[41] Weizhe Chen, Zihan Zhou, Yi Wu, and Fei Fang. Temporal induced self-play for stochastic bayesian games. *arXiv preprint arXiv:2108.09444*, 2021.

[42] Hanbaek Lyu. Convergence and complexity of block coordinate descent with diminishing radius for nonconvex optimization. *arXiv preprint arXiv:2012.03503*, 2020.

[43] Emrah Demir, Tolga Bektas, and Gilbert Laporte. An adaptive large neighborhood search heuristic for the pollution-routing problem. *Eur. J. Oper. Res.*, 223(2):346–359, 2012.

[44] Filippos Christianos, Georgios Papoudakis, Arrasy Rahman, and Stefano V. Albrecht. Scaling multi-agent reinforcement learning with selective parameter sharing. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 1989–1998. PMLR, 2021.

[45] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[46] Zifan Wu, Chao Yu, Deheng Ye, Junge Zhang, Hankz Hankui Zhuo, et al. Coordinated proximal policy optimization. *Advances in Neural Information Processing Systems*, 34:26437–26448, 2021.