

---

# A Structured Prediction Approach for Generalization in Cooperative Multi-Agent Reinforcement Learning

---

**Nicolas Carion**  
Facebook, Paris  
Lamsade, Univ. Paris Dauphine  
alcinos@fb.com

**Gabriel Synnaeve**  
Facebook, NYC  
gab@fb.com

**Alessandro Lazaric**  
Facebook, Paris  
lazaric@fb.com

**Nicolas Usunier**  
Facebook, Paris  
usunier@fb.com

## Abstract

Effective coordination is crucial to solve multi-agent collaborative (MAC) problems. While centralized reinforcement learning methods can optimally solve small MAC instances, they do not scale to large problems and they fail to generalize to scenarios different from those seen during training. In this paper, we consider MAC problems with some intrinsic notion of locality (e.g., geographic proximity) such that interactions between agents and tasks are locally limited. By leveraging this property, we introduce a novel structured prediction approach to assign agents to tasks. At each step, the assignment is obtained by solving a centralized optimization problem (the inference procedure) whose objective function is parameterized by a learned scoring model. We propose different combinations of inference procedures and scoring models able to represent coordination patterns of increasing complexity. The resulting assignment policy can be efficiently learned on small problem instances and readily reused in problems with more agents and tasks (i.e., zero-shot generalization). We report experimental results on a toy search and rescue problem and on several target selection scenarios in StarCraft<sup>®</sup>: Brood War<sup>®</sup><sup>1</sup>, in which our model significantly outperforms strong rule-based baselines on instances with 5 times more agents and tasks than those seen during training.

## 1 Introduction

Multi-agent collaboration (MAC) problems often decompose into several intermediate tasks that need to be completed to achieve a global goal. A common measure of size, or difficulty, of MAC problems is the number of agents and tasks: more tasks usually require longer-term planning, the joint action space grows exponentially with the number of agents, and the joint state space is exponential in both the numbers of tasks and agents. While general-purpose reinforcement learning (RL) methods [29] are theoretically able to solve (centralized) MAC problems, their learning (e.g., estimating the optimal action-value function) and computational (e.g., deriving the greedy policy from an action-value function) complexity grows exponentially with the dimension of the problem. A way to address this limitation is to learn in problems with few agents and a small planning horizon and then *generalize* the solution to more complex instances. Unfortunately, standard RL methods are not able to perform any meaningful generalization to scenarios different from those seen during training. In this paper

---

<sup>1</sup>StarCraft and Brood War are trademarks or registered trademarks of Blizzard Entertainment, Inc., in the U.S. and/or other countries. Nothing in this Paper should be construed as approval, endorsement, or sponsorship by Blizzard Entertainment, Inc.

we study problems whose structure can be exploited to learn policies in small instances that can be efficiently generalized across scenarios of different size.

Well-known MAC problems that are solved by a suitable sequence of agent-task assignments include search and rescue, predator-prey problems, fleet coordination, or managing units in video games. In all these problems, the dynamics describing the interaction of the “objects” in the environment (i.e., agents and tasks) is regulated by *constraints* that may greatly simplify the problem. A typical example is *local proximity*, where objects’ actions may only affect nearby objects (e.g., in the predator-prey, the prey’s movements only depend on nearby agents). Similarly, constraints may be related to *assignment proximity*, as agents may only interact with agents assigned to the same task.

The structure of problems with *constrained interaction* has been exploited to simplify the learning of value functions [e.g., 23] or dynamics of the environment [e.g., 12]. These approaches effectively generalize from easier to more difficult instances: we may train on small environments where the sample complexity is practical and generalize to large problems without ever training on them (*zero-shot generalization*). The main drawback is that when generalizing value functions or the dynamics, the optimal (or greedy) policy still needs to be recomputed at each new instance, which usually requires solving an optimization problem with complexity exponential in the number of objectives (e.g., maximizing the action-value function over the joint action space).

In this paper, we build on the observation that in MAC problems with constrained interaction, optimal policies (or good approximations) can be effectively represented as a combination of coordination patterns that can be expressed as reactive rules, such as creating subgroups of agents to solve a single task, avoiding redundancy, or combinations of both. We decompose agents’ policies into a high-level agent-task assignment policy and a low-level policy that prescribes the actual actions agents should take to solve the assigned task. As the most critical aspect of MAC problems is the coordination between agents, we assume low-level policies are provided in advance and we focus on learning effective high-level policies. To leverage the structure of the assignment policy, we propose a structured prediction approach, where agents are assigned to tasks as a result of an optimization problem. In particular, we distinguish between the *coordination inference procedure* (i.e., the optimization problem itself) and *scoring models* (the objective function) that provide a score to agent-task and task-task pairs. In its more complex instance, we define a quadratic inference procedure with linear constraints, where the objective function uses learned pairwise scores between agents and tasks for the linear part of the objective, and between different tasks for the quadratic part. With this structure we address the intrinsic exponential complexity of learning in large MAC problems through *zero-shot generalization*: **1)** the parameters of the scoring model can be learned in small instances, thus keeping the learning complexity low, **2)** the coordination inference procedure can be generalized to an arbitrary number of agents and tasks, as its computational complexity is polynomial in the number of agents and tasks. We study the effectiveness of this approach on a search and rescue problem and different battle scenarios in “StarCraft<sup>®</sup>: Brood War<sup>®</sup>”. We show that the linear part of the optimization problem (i.e., using agent-task scores) represents simple coordination patterns such as assigning agents to their closest tasks, while the quadratic part (i.e., using task-task scores) may capture longer-term coordination such as spreading the different agents to tasks that are far away to each other or, on the contrary, create groups of agents that focus on a single task.

## 2 Related Work

Multi-agent reinforcement learning has been extensively studied, mostly in problems of decentralized control and limited communication (see Busoniu et al. [2] for a survey). By contrast, this paper focuses on *centralized control under full state observation*.

Our work is closely related to generalization in relational Markov Decision Processes [12] and decomposition approaches in loosely and weakly coupled MDPs [27, 20, 11, 30, 23]. The work on relational MDPs and the related object-oriented MDPs and first-order MDPs [12, 6, 25] focus on learning and planning in environments where the state/action space is compactly described in terms of objects (e.g., agents) that interact with each other, without prior knowledge of the actual number of objects involved. Most of the work in this direction is devoted to either efficiently estimating the environment dynamics, or approximate the planning in new problem instances. Whereas the type of environments and problems we aim at are similar, we focus here on model-free learning of policies that generalize to new (and larger) problem instances *without* replanning.

Loosely or weakly coupled MDPs are another form of structured MDPs, which decompose into smaller MDPs with nearly independent dynamics. These works mostly follow a decomposition approach in which global action-value functions are broken down into independent parts that are either learned individually, or serve as guide for an effective parameterization for function approximation. The policy parameterization we develop follows the task decomposition approach of Proper and Tadepalli [23], but the policy structures we propose are different. Proper and Tadepalli [23] develop policies based on pairwise interaction terms between tasks and agents similar to our quadratic model, but the pairwise terms are based on interactions dictated by the dynamics of the environment (e.g., agent actions that directly impact the effect of other actions) aiming at a better estimation of the value function of low-level actions of the agents once an assignment is fixed, whereas our quadratic term aims at assessing the long-term value of an assignment.

Many deep reinforcement learning algorithms have been recently proposed to solve MAC problems with a variable number of agents, using different variations of communication and attention over graphs [28, 7, 36, 35, 15, 14, 19, 26]. However, most of these algorithms focus on fixed-size action spaces, and little evidence has been given that these approaches generalize to larger problem instances [32, 8, 36]. Rashid et al. [24] and Lin et al. [18] address the problem of learning (deep) decentralized policies with a centralized critic during learning in structured environments. While they do not address the problem of generalization, nor the problem of learning a centralized controller, we use their idea of a separate critic computed based on the full state information during training.

### 3 Multi-agent Task Assignment

We formalize a general MAC problem. To keep notation simple, we present a fixed-size description, but the end goal is to design policies that can be applied to environments of arbitrary size.

As customary in reinforcement learning, the objective of solving the tasks is encoded through a reward function that needs to be maximized over the long run by the coordinated actions of all agents. An environment with  $m$  tasks and  $n$  agents is modeled as an MDP  $\langle \mathcal{S}^m, \mathcal{X}^n, \mathcal{A}^n, r, p \rangle$ , where  $\mathcal{S}$  is the set of possible states of each *task* (indexed by  $j = 1, \dots, m$ ),  $\mathcal{X}$  and  $\mathcal{A}$  are the set of states and actions of each agent (indexed by  $i = 1, \dots, n$ ). We denote the joint states/actions by  $\mathbf{s} \in \mathcal{S}^m$ ,  $\mathbf{x} \in \mathcal{X}^n$ , and  $\mathbf{a} \in \mathcal{A}^n$ . The reward function is defined as  $r : \mathcal{S}^m \times \mathcal{X}^n \times \mathcal{A}^n \rightarrow \mathbb{R}$  and the stochastic dynamics is  $p : \mathcal{S}^m \times \mathcal{X}^n \times \mathcal{A}^n \rightarrow \Delta(\mathcal{S}^m \times \mathcal{X}^m)$ , where  $\Delta$  is the probability simplex over the (next) joint state set. A joint deterministic policy is defined as a mapping  $\pi : \mathcal{S}^m \times \mathcal{X}^n \rightarrow \mathcal{A}^n$ . We consider the episodic discounted setting where the action-value function is defined as  $Q^\pi(\mathbf{s}, \mathbf{x}, \mathbf{a}) = \mathbb{E}_\pi [r(\mathbf{s}, \mathbf{x}, \mathbf{a}) + \sum_{t=1}^T \gamma^t r(\mathbf{s}_t, \mathbf{x}_t, \mathbf{a}_t)]$ , where  $\gamma \in [0, 1)$ ,  $\mathbf{a}_t = \pi(\mathbf{s}_t, \mathbf{x}_t)$  for all  $t \geq 1$ ,  $\mathbf{s}_t$  and  $\mathbf{x}_t$  are sampled from  $p$ , and  $T$  is the time by when all tasks have been solved. The goal is to learn a policy  $\pi$  close to the optimal  $\pi^* = \arg \max_\pi Q^\pi$  that we can easily generalize to larger environments.

**Task decomposition.** Following a similar task decomposition approach as [30] and [23], we consider hierarchical policies that first assign each agent to a task, and where actions are given by a lower-level policy that only depends on the state of individual agents and the task they are assigned to. Denoting by  $\mathcal{B} = \{\beta \in \{0, 1\}^{n \times m} : \sum_{j=1}^m \beta_{ij} = 1\}$  the set of assignment matrices of agents to tasks, an assignment policy first chooses  $\hat{\beta}(\mathbf{s}, \mathbf{x}) \in \mathcal{B}$ . In the second step, the action for each agent is chosen according to a lower-level policy  $\tilde{\pi}$ . Using  $\pi_i(\mathbf{s}, \mathbf{x})$  to denote the action of agent  $i$  and  $\hat{\beta}_i(\mathbf{s}, \mathbf{x}) \in \{1, \dots, m\}$  for the task assigned to agent  $i$ , we have  $\pi_i(\mathbf{s}, \mathbf{x}) = \tilde{\pi}(s_{\hat{\beta}_i(\mathbf{s}, \mathbf{x})}, x_i)$ , where  $s_j$  and  $x_i$  are respectively the internal states of task  $j$  and agent  $i$  in the full state  $(\mathbf{s}, \mathbf{x})$ . In the following, we focus on learning high-level assignment policies responsible for the collaborative behavior, while we assume that the lower-level policy  $\tilde{\pi}$  is *known and fixed*.

### 4 A Structured Prediction Approach

In this section we introduce a novel method for centralized coordination. We propose a structured prediction approach in which the agent-task assignment is chosen by solving an optimization problem. Our method is composed of two components: a **coordination inference procedure**, which defines the shape of the optimization problem and thus the type of coordination between agents and tasks, and a **scoring model**, which receives as input the state of agents and tasks and returns the parameters of the objective function of the optimization. The combination of these two components defines an agent-task assignment policy  $\hat{\beta}$  that is then passed to the low-level policy  $\tilde{\pi}$  (that we assume fixed) which returns the actual actions executed by the agents. Finally, we use a **learning algorithm** to

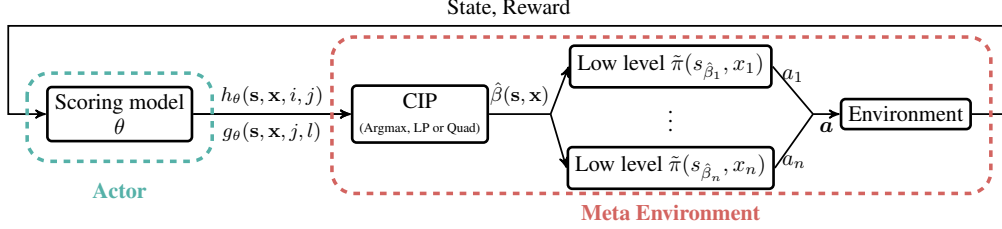


Figure 1: Illustration of the approach, where the agent-task assignment is computed by a coordination inference procedure (CIP) which receives as input agent-task ( $h$ ) and task-task ( $g$ ) scores computed by a scoring model parametrized by  $\theta$ . The assignment  $\hat{\beta}$  is then passed to fixed low level policies that return the actions played by each agent. The learning algorithm tunes  $\theta$  and performs “meta-actions”  $h_\theta$  and  $g_\theta$  on to the “meta-environment” composed by the inference procedure, the low-level policies, and the actual environment.

learn the parameters of the scoring model itself in order to maximize the performance of  $\hat{\beta}$ . The overall scheme of this method is illustrated in Fig. 1.

#### 4.1 Coordination Inference Procedures

The collaborative behaviors that we can represent are tied to the specific form of the objective function and its constraints. The formulations we propose are motivated by collaboration patterns important for long-term performance, such as creating subgroups of agents, or spreading agents across tasks.

**Greedy assignment.** The simplest form of assignment is to give a score to each agent-task pair and then assign each agent to the task with the highest score, ignoring other agents at inference time. In this approach, that we refer to as AMAX strategy, a model  $h_\theta(\mathbf{s}, \mathbf{x}, i, j) \in \mathbb{R}$  parameterized by  $\theta$  receives as input the full state and returns the score of agent  $i$  for task  $j$ . The associated policy is then

$$\hat{\beta}^{\text{AMAX}}(\mathbf{s}, \mathbf{x}, \theta) = \arg \max_{\beta \in \mathcal{B}} \sum_{i,j} \beta_{i,j} h_\theta(\mathbf{s}, \mathbf{x}, i, j), \quad (1)$$

which corresponds to assigning each agent  $i$  to the task  $j$  with largest score  $h_\theta(\mathbf{s}, \mathbf{x}, i, j)$ . As a result, the complexity of coordination is reduced from  $O(m^n)$  (i.e., considering all possible agent-to-task assignments) down to a linear complexity  $O(nm)$  (once the function  $h_\theta$  has been evaluated on the full state). We also notice that AMAX bears strong resemblance to the strategy used in [23], where the scores are replaced by approximate value functions computed for any agent-task pair.<sup>2</sup>

**Linear Program assignment.** Since AMAX ignores interactions between agents, it tends to perform poorly in scenarios where a task has a high score for all agents (i.e.,  $h(\mathbf{s}, \mathbf{x}, i, j)$  is large for a given  $j$  and for all  $i$ ). In this situation, all agents are assigned to the same task, implicitly assuming that the “value” of solving a task is additive in the number of agents assigned to it (i.e., if  $n$  agents are assigned to the same task then we could collect a reward  $n$  times larger). While this may be the case when the number of agents assigned to the same task is small, in many practical scenarios this effect tends to saturate as more agents are assigned to a single task. A simple way to overcome this undesirable behavior is to impose a restriction on the number of agents assigned to a task. We can formalize this intuition by introducing  $\mu_{(i,j)}(\mathbf{s}, \mathbf{x})$  as the *contribution* of an agent  $i$  to a given task  $j$ , and  $u_j(\mathbf{s}, \mathbf{x})$  as the *capacity* of the task  $j$ . In the simplest case, we may know the maximum number of agents  $n_j$  that is necessary to solve each task  $j$ , and we can set the capacity of each task to be  $n_j$ , and all the contributions  $\mu_{i,j}$  to be 1. Depending on the problem, the capacities and contributions are either prior knowledge or learned as a function of the state. Formally, denoting by  $\bar{\mathcal{B}}(\mathbf{s}, \mathbf{x})$  the constrained assignment space

$$\bar{\mathcal{B}}(\mathbf{s}, \mathbf{x}) = \left\{ \beta \in \{0, 1\}^{n \times m} \mid \forall i, \sum_{j=1}^m \beta_{i,j} \leq 1; \forall j, \sum_{i=1}^n \mu_{i,j}(\mathbf{s}, \mathbf{x}) \beta_{i,j} \leq u_j(\mathbf{s}, \mathbf{x}) \right\}, \quad (2)$$

the resulting policy infers the assignment by solving an integer linear program

$$\hat{\beta}^{\text{LP}}(\mathbf{s}, \mathbf{x}, \theta) = \arg \max_{\beta \in \bar{\mathcal{B}}(\mathbf{s}, \mathbf{x})} \sum_{i,j} \beta_{i,j} h_\theta(\mathbf{s}, \mathbf{x}, i, j), \quad (3)$$

<sup>2</sup>An alternative approach is to sample assignments proportionally to  $h_\theta(\mathbf{s}, \mathbf{x}, i, j)$ . Preliminary empirical tests of this procedure performed worse than AMAX and thus we do not report its results.

Notice that even with the additional constraints in (2), some agents may not be assigned to any task, hence inequality  $\sum_{j=1}^m \beta_{i,j} \leq 1$  instead of strict equality.

In order to optimize (3) efficiently, we trade off accuracy for speed by solving its linear relaxation using an efficient LP library [1], and retrieving a valid assignment using greedy rounding: Let us denote as  $\beta_{i,j}^*$  the solution of the relaxed ILP; we iterate over agents  $i$  in descending order of  $\max_j \beta_{i,j}^*$ , and assign each agent to the task of maximum score that is not already saturated.

**Quadratic Program assignment.** The linear program above avoids straightforward drawbacks of a greedy assignment policy, but is unable to represent grouping patterns that are important on the long-run in coordination and collaboration problems. For instance, it may be convenient to “spread” agents among unrelated tasks, or, on the contrary, group agents together on a single task (up to the constraints) and then move to other tasks in a sequential fashion. Such grouping patterns can be well represented with a quadratic objective function of the form

$$\hat{\beta}^{\text{QUAD}}(\mathbf{s}, \mathbf{x}, \theta) = \arg \max_{\beta \in \bar{\mathcal{B}}(\mathbf{s}, \mathbf{x})} \left[ \sum_{i,j} \beta_{i,j} h_{\theta}(\mathbf{s}, \mathbf{x}, i, j) + \sum_{i,j,k,l} \beta_{i,j} \beta_{k,l} g_{\theta}(\mathbf{s}, \mathbf{x}, j, l) \right], \quad (4)$$

where  $g_{\theta}(\mathbf{x}, \mathbf{s}, j, l)$  plays the role of a (signed) distance between two tasks and  $\bar{\mathcal{B}}$  is the same set of constraints as in (2). In the extreme case where  $g_{\theta}(\mathbf{x}, \mathbf{s}, \dots)$  is a diagonal matrix, the quadratic part of the objective favors agents to carry on the same task (if the diagonal terms are positive) or on the contrary carry on different tasks (if the terms are negative). In general, negative  $g_{\theta}(\mathbf{x}, \mathbf{s}, j, l)$  disfavors agents to be assigned to  $j$  and  $l$  at the same time step depending on  $|g_{\theta}(\mathbf{x}, \mathbf{s}, j, l)|$ . For instance, in the search and rescue problem, this captures the idea that agents should spread to explore the map.

As for the LP, we optimize a continuous relaxation of (4) using the same rounding procedure. The objective function may not be concave, because there is no reason for  $g_{\theta}(\mathbf{x}, \mathbf{s}, \dots)$  to be negative semi-definite. In practice, we use the Frank-Wolfe algorithm [9] to deal with the linear constraints; the algorithm is guaranteed to converge to a local maximum and was efficient in our experiments.

## 4.2 Scoring Models

In order to allow generalizing the coordination policy  $\hat{\beta}$  to instances of different size, the  $h_{\theta}$  and  $g_{\theta}$  functions should be able to compute scores for pairs agents/tasks and tasks/tasks, independently of the actual amount of those. In order to make the presentation concrete, in the following we illustrate different scoring models in the case where the agents and tasks are objects located in a fixed-size 2D grid, and are characterized by an internal state. The position on the grid is part of this internal state.<sup>3</sup>

**Direct Model (DM).** The first option is to use a fully decomposable approach (*direct model*), where the score for the pair  $(i, j)$  only depends on the internal states of agent  $i$  and task  $j$ :  $h_{\theta}(\mathbf{s}, \mathbf{x}, i, j) = \tilde{h}_{\theta}(s_j, x_i)$  for some function  $\tilde{h} : \mathcal{S} \times \mathcal{X} \rightarrow \mathbb{R}$ . This model only uses the features of the pair of objects to compute the score. Precisely,  $\tilde{h}_{\theta}(s_j, x_i)$  is obtained by concatenating the feature vectors of agent  $i$  and task  $j$ , and by feeding them to a fully-connected network of moderate depth. In the quadratic program strategy, the function  $g_{\theta}$  follows the same structure as  $h$  (but uses different weights).

While this approach is computationally efficient, if used in the simple AMAX procedure (1), it leads to a policy that ignores interactions between agents altogether and is thus unable to represent effective collaboration patterns. As a result, the direct model should be paired with more sophisticated inference procedures to achieve more complex coordination patterns. On the other hand, as it computes scores by ignoring surrounding agents and tasks, once learned on small instances, it can be directly applied (i.e., zero-shot generalization) to larger instances independently of the number of agents and tasks.

**General Model.** An alternative approach is to take  $h_{\theta}$  as a highly expressive function of the full state. The main challenge in this case is to define an architecture that can output scores for a variable number of agents and tasks. In the case where agents/tasks are in a 2D grid, we can define a **positional embedding model (PEM)** (see App. A for more details) that computes scores following ideas similar to non-local networks [33]. We use a deep convolutional neural network that outputs  $k$  feature planes at the same resolution as the input. This implies that each cell is associated with  $k$  values that we treat as an embedding of the position. We divide this embedding in two sub-embeddings of size  $k/2$ ,

<sup>3</sup>Notice that the direct model illustrated below does not leverage this specific scenario, which, on the other hand, is needed to define the general model.

to account for the two kinds of entities: the first  $k/2$  values represent an embedding of an agent, and the remaining ones represent an embedding of a task. To compute the score between two entities, we concatenate the embeddings of both entities and the input features of both of them, and run that through a fully connected model, using the same topology as described for the direct model.

By leveraging the *full* state, this model can capture non-local interactions between agents and tasks (unlike the direct model) depending on the receptive field of the convolutional network. Larger receptive fields allow the model to learn more sophisticated scoring functions and thus better policies. Furthermore, it can be applied to variable number of agents and tasks as a position contains at most one agent and one task. Nonetheless, as it depends on the full state, the application to larger instances means that the model may be tested on data points outside the support of the training distribution. As a result, the scores computed on larger instances may be not accurate, thus leading to policies that can hardly generalize to more complex instances.

### 4.3 Learning Algorithm

As illustrated in Fig. 1, the learning algorithm optimizes a policy parametrized by  $\theta$  that returns as actions the scores  $h_\theta$  and  $g_\theta$ , while the combination of the assignment  $\hat{\beta}$  returned by the optimization, the low-level policy  $\tilde{\pi}$  and the environment, plays the role of a “meta-environment”. While any policy gradient algorithm could be used to optimize  $\theta$ , in the experiments we use a synchronous Advantage-Actor-Critic algorithm [21], which requires computing a state-value function. As advocated by Rashid et al. [24] in the context of learning decentralized policies, we use a global value function that takes the whole state as input. We use a CNN similar to the one of PEM, followed by a spatial pooling and a linear layer to output the value. This value function is used only during training, hence its parametrization does not impact the potential generalization of the policy (more details in App. B).

**Correlated exploration.** Reinforcement learning requires some form of exploration scheme. Many algorithms using decompositional approaches for MAC problems [11, 30, 23, 24] rely on variants of Q-learning or SARSA and directly randomize the low-level actions taken by the agents. However, this approach is not applicable to our framework. In our case, the randomization is applied to the scores (denoted as  $H_\theta(\mathbf{s}, \mathbf{x}, i, j)$  and  $G_\theta(\mathbf{s}, \mathbf{x}, j, l)$ ) before passing them to the inference procedure. We can’t use a simple gaussian noise, since at the beginning of the training, when the scoring model is random, it would cause the agents to be assigned to different tasks at each step, thus preventing them from solving any task and getting any positive reward. To alleviate this problem, we correlate temporally the consecutive realizations of  $H_\theta$  and  $G_\theta$  using auto-correlated noise as studied in [e.g., 34, 17], so that the actual sequence of assignments executed by the agent is also correlated. To correlate the parameters over  $p$  steps, at time  $t$ , we sample  $H_{t,\theta}(i, j)$  according to (dropping dependence on  $(\mathbf{s}_t, \mathbf{x}_t)$  for clarity):  $\mathcal{N}(h_{t,\theta}(i, j) + \sum_{t'=t-p}^{t-1} (H_{t',\theta}(i, j) - h_{t',\theta}(i, j)), \frac{\sigma}{p})$ . This is equivalent to correlating the sampling noise over a sliding window of size  $p$ . During the update of the model, we ignore the correlation, and assume that the actions were sampled according to  $\mathcal{N}(h_{t,\theta}(i, j), \sigma)$ .

## 5 Experiments

We report results in two different problems: search and rescue and target selection in StarCraft. Both experiments are designed to test the generalization performance of our method: we learn the scoring models on small instances and the learned policy is tested on larger instances with no additional re-training. We test different combinations of coordination inference procedures and scoring models. Among the inference procedures, AMAX should be considered as a basic baseline, while we expect LP to express some interesting coordination patterns. The QUAD is expected to achieve the better performance in the training instance, although its more complex coordination patterns may not effectively generalize well to larger instances. Among the scoring models, PEM should be able to capture dependencies between agents and tasks in a single instance but may fail to generalize when tested on instances with a number of agents and tasks not seen at training time. On the other hand, the simpler DM should generalize better if paired with a good coordination inference procedure.

The PEM + AMAX combination roughly corresponds to independent A2C learning and can be seen as the standard approach baseline, and we also provided strong hand-crafted baselines. Most previous approaches didn’t aim achieving effective generalization, and often relied on fixed-size action spaces, rendering direct comparison impractical.

Table 1: *Search and Rescue*. Average number of steps to solve the validation episodes, depending on the train scenario.  $\Delta$  denotes the improvement over baseline. Best results are in bold, with an asterisk when they are statistically ( $p < 0.0001$ ) better than the second best. Results like "10.3(1.1%)" mean that the evaluation failed in 1.1% of the test scenarios, and had an average score of 10.3 on the remaining 98.9%. In case of evaluation failures, the reported improvement over baseline are indicative (reported in italics between parenthesis).

	Train ( $n \times m$ )	Test	Baseline	Topline	AMAX-PEM	LP-PEM	QUAD-PEM	AMAX-DM	LP-DM	QUAD-DM
lower is better	$2 \times 4$	$2 \times 4$	14.34	10.28	11.98	12.09	<b>11.44</b>	13.78	11.98	11.55
		$5 \times 10$	13.61	7.19	13.36	10.69	9.67	12.49	10.24	<b>9.32*</b>
		$8 \times 15$	11.8	n.a	<i>15.8(0.7%)</i>	9.86	<i>10.3(1.1%)</i>	11.06	9.71	<b>7.85*</b>
lower is better	$5 \times 10$	$2 \times 4$	14.34	10.28	12.05	12.94	<i>13.23(1%)</i>	13.84	12.22	<b>11.78*</b>
		$5 \times 10$	13.61	7.19	9.84	10.24	10.43	12.26	10.12	<b>9.36*</b>
		$8 \times 15$	11.8	n.a	8.60	9.37	9.51	10.57	8.63	<b>7.95*</b>
higher is better	In domain $\Delta$				22%	20%	21%	7%	21%	<b>25%</b>
	Out of domain $\Delta$				(22%)	17%	(18%)	7%	21%	<b>29%</b>
	Total $\Delta$		0%	38%	(18%)	18%	(19%)	7%	21%	<b>28%</b>

## 5.1 Search and Rescue

**Setting.** We consider a search and rescue problem on a grid environment of 16 by 16 cells. Each instance is characterized by a set of  $n$  ambulances (i.e., agents) and  $m$  victims (i.e., tasks). The goal is that all the victims are picked up by one of the ambulances as quickly as possible. This problem can be seen as a Multi-vehicle Routing Problem (MVR), which makes it NP-hard.

The reward is  $-0.01$  per time-step until the end of an episode (when all the victims have been picked up). The learning task is challenging because the reward is uninformative and coupled; it is difficult for an agent to assign credit to the solution of an individual tasks (i.e., picking up a victim). The assignment policy  $\hat{\beta}$  matches ambulances to victims, while the low-level policy  $\tilde{\pi}$  takes an action to reduce the distance between the ambulance and its assigned victim. In this environment, only one ambulance is needed to pick-up a particular victim, hence the saturation  $u_j(s, \mathbf{x})$  is set to 1. We trained our models on two instances ( $n = 2, m = 4$  and  $n = 5, m = 10$ ) and we test it on the trained scenarios, as well as in instances with a larger number of victims and ambulances. At test time, we evaluate the policies on a fixed set of 1000 random episodes (with different starting positions). The agents use the same variance and number of correlated steps as they had during training. The results are summarized in Tab. 1, where we report the average number of steps required to complete the episodes. We also report the results of a greedy baseline policy that always assigns each ambulance to the closest victim, and a topline policy that solves each instance optimally (see App. C.3 for more details). Because of its computational cost, the topline for the biggest instance ( $8 \times 15$ ) is not available. In the last rows of the table, we aggregate the average improvements over the baseline ( $100 * \frac{\text{baseline} - \text{method}}{\text{baseline}}$ ). The in-domain scores correspond to the scores obtained when the test instance matches the train instance. Conversely, the out of domain scores correspond to the performances on unseen instances. Note that no model was trained on  $8 \times 15$ .

**Results.** Firstly, the PEM scoring model tends to overfit to the train scenario, leading to poor generalization (i.e., in some configuration it fails to solve the problem). On the other hand, for the DM, the generalization is very stable. Regarding the inference procedures<sup>4</sup>, AMAX tends to perform at least as well as the greedy baseline, by learning how to compute the relevant distance function between an ambulance and a victim. The LP strategy can rely on the same distance function and perform better, by since it enforces the coordination and avoids sending more than one ambulance to the same victim. Finally, the QUAD strategy is able to learn long-term strategies, and in particular how to spread efficiently the ambulances across the map (e.g., if two victims are very close, it is wasteful to assign two distinct ambulances to them, since one can efficiently pick-up both victims sequentially, while the other ambulance deals with further victims) (see App.C.5 for more discussion).

## 5.2 Target Selection in StarCraft<sup>®</sup>

**Setting.** We focus on a specific sub-problem in StarCraft<sup>®</sup>: battles between two groups of units. This setting, often referred to as *micromanagement*, has already been studied in the literature, using a mixture of scripted behaviours and search [4, 3, 22, 5], or using RL techniques [32, 8]. In these

<sup>4</sup>We study their performance when paired with DM.

battles, a crucial aspect of the policy is to assign a target enemy unit (*the task*) to each of our units (*the agents*), in a coordinated way. Since we focus on the agent-task assignment (the high-level policy  $\hat{\beta}$ ), we use a simple low-level policy for the agents (neither learnt nor scripted) relying on the built-in “attack” command of the game, which moves each unit towards its target and shoots as soon as the target is in range. This contrasts with previous works, which usually allow more complex movement patterns (e.g., retreating while the weapon is reloading). While such low-level policies could be integrated in our framework, we preferred to use the simplest “attack” policy to better assess the impact of the high-level coordination.

In this problem, the capacity  $u_j(\mathbf{s}, \mathbf{x})$  of a task  $j$  is defined as the remaining health of the enemy unit, and the contribution  $\mu_{i,j}(\mathbf{s}, \mathbf{x})$  of an agent  $i$  to this task is defined as the amount of damage dealt by unit  $i$  to the enemy  $j$ . These constraints are meant to avoid dealing more damage to an enemy than necessary to kill it, a phenomenon known as *over-killing*.

Given the poor results of PEM in the previous experiment, we only train DM with all the possible inference procedures. Each unit is represented by its features: whether it is an enemy, its position, velocity, current health, range, cool-down (number of frames before the next possible attack), and one hot encoding of its type. This amounts to 8 to 10 features per units, depending on the scenario. For training, we sample 100 sets of hyper-parameters for each combination of model/scenario, and train them for 8 hours on a Nvidia Volta. In this experiment, we found that the training algorithm is relatively sensitive to the random seed. To better assess the performances, we re-trained the best set of hyper-parameters for each model/scenario on 10 random seeds, for 18 hours. The performances we report are the median of the performances of all the seeds, to alleviate the effects of the outliers. The results are aggregated in Tab. 2. Although the number of units is a good indicator of the difficulty of the environment, whether the numbers of units are balanced in both teams dramatically change the “dynamics” of the game. For instance, zh10v12 is unbalanced and thus much more difficult than zh11v11, which is balanced. The performance of the baseline can be seen as a relatively accurate estimate of the difficulty of the scenario. See App. D.2 for a description of the heuristics used for comparison, and App. D.1 for a description of the training scenarios. We also provide a more detailed results in App. D.5.

**Results.** As StarCraft<sup>®</sup> is a real-time game, one first concern regards the runtime of our algorithm. In the biggest experiment, involving 80 units vs 82, our algorithm returned actions in slightly more than 500ms (5ms for the forward in the model, 500ms to solve the inference of QUAD). Given the frequency at which we take actions (every 6 frames), such timings allow real-time play in StarCraft<sup>®</sup>.

Amongst the scenarios, the Wraith setting ( $wNvM$ ) are the ones where the assumption of independence between the tasks holds the best, since in this case there are no collisions between units. These scenarios also require good coordination, since it is important to focus fire on the same unit. During these battles, both armies tend to overlap totally, hence it becomes almost impossible to use surrogate coordination principles such as targeting the closest unit. In this case, the quadratic part of the score function is crucial to learn focus-firing and results show that without the ability to represent such a long-term coordination pattern, both LP and AMAX fail to reach the same level of performance. Notably, the coordination pattern learned by QUAD generalizes well, outperforming the best heuristics in instances as much as 5 times the size of the training instance.

The other setting, Marine ( $mNvM$ ) and Zergling-Hydralisk ( $zhNvM$ ) break the independence assumption because the units now have collisions. It is even worse for the Zerglings, since they are melee units. The coordination patterns are then harder to learn for the QUAD model, and they generalize poorly. However, these scenarios with collisions also tend to require less long-term

Table 2: StarCraft<sup>®</sup>. Average win-rate of different methods (best in bold). See confidence intervals in the full table in App D.5

Train	Test	Best heuristic	LP	QUAD	AMAX
m10v10	m5v5	0.88	<b>0.90</b>	0.83	0.84
	m10v10	0.77	<b>0.94</b>	0.83	0.82
	m10v11	0.25	<b>0.52</b>	0.28	0.29
	m15v15	0.75	<b>0.92</b>	0.69	0.77
	m15v16	0.40	<b>0.68</b>	0.32	0.43
	m30v30	0.69	<b>0.74</b>	0.06	0.36
w15v17	w15v17	0.81	0.53	<b>0.89</b>	0.30
	w30v34	0.90	0.76	<b>0.99</b>	0.37
	w30v35	0.60	0.56	<b>0.94</b>	0.24
	w60v67	0.07	0.33	<b>0.72</b>	0.13
	w60v68	0.01	0.21	<b>0.52</b>	0.07
	w80v82	0.32	0.11	<b>0.36</b>	0.03
zh10v10	zh10v10	0.86	<b>0.90</b>	0.83	0.84
	zh10v11	0.30	<b>0.46</b>	0.24	0.40
	zh10v12	0.03	<b>0.06</b>	0.01	<b>0.06</b>
	zh11v11	<b>0.87</b>	<b>0.87</b>	0.75	0.80
	zh12v12	<b>0.85</b>	0.82	0.64	0.75



coordination, and the immediate coordination patterns learned by the LP model are enough to significantly outperform the heuristics, even when transferring to unseen instances.

## **6 Conclusion**

In this paper we proposed a structured approach to multi-agent coordination. Unlike previous work, it uses an optimization procedure to compute the assignment of agents to tasks and define suitable coordination patterns. The parameterization of this optimization procedure is seen as the continuous output of an RL trained model. We showed on two challenging problems the effectiveness of this method, in particular in generalizing from small to large instance.

## References

- [1] The glop linear solver. <https://developers.google.com/optimization/lp/glop>.
- [2] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, And Cybernetics-Part C: Applications and Reviews*, 38 (2), 2008, 2008.
- [3] David Churchill and Michael Buro. Portfolio greedy search and simulation for large-scale combat in StarCraft. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pages 1–8. IEEE, 2013.
- [4] David Churchill, Abdallah Saffidine, and Michael Buro. Fast heuristic search for RTS game combat scenarios. In *AIIDE*, pages 112–117, 2012.
- [5] David Churchill, Zeming Lin, and Gabriel Synnaeve. An analysis of model-based heuristic search techniques for StarCraft combat scenarios. AAAI Publications, Thirteenth Artificial Intelligence and Interactive . . . , 2017.
- [6] Carlos Diuk, Andre Cohen, and Michael L Littman. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pages 240–247. ACM, 2008.
- [7] Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2137–2145. Curran Associates, Inc., 2016.
- [8] Jakob N Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [9] Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval Research Logistics (NRL)*, 3(1-2):95–110, 1956.
- [10] Ambros Gleixner, Leon Eifler, Tristan Gally, Gerald Gamrath, Patrick Gemander, Robert Lion Gottwald, Gregor Hendel, Christopher Hojny, Thorsten Koch, Matthias Miltenberger, Benjamin Müller, Marc E. Pfetsch, Christian Puchert, Daniel Rehfeldt, Franziska Schläpfer, Felipe Serrano, Yuji Shinano, Jan Merlin Viernickel, Stefan Vigerske, Dieter Weninger, Jonas T. Witt, and Jakob Witzig. The scip optimization suite 5.0. Technical Report 17-61, ZIB, Takustr. 7, 14195 Berlin, 2017.
- [11] Carlos Guestrin, Michail Lagoudakis, and Ronald Parr. Coordinated reinforcement learning. In *ICML*, volume 2, pages 227–234, 2002.
- [12] Carlos Guestrin, Daphne Koller, Chris Gearhart, and Neal Kanodia. Generalizing plans to new environments in relational mdps. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence, IJCAI’03*, pages 1003–1010, San Francisco, CA, USA, 2003. Morgan Kaufmann Publishers Inc.
- [13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [14] Jiechuan Jiang and Zongqing Lu. Learning attentional communication for multi-agent cooperation. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 7265–7275. Curran Associates, Inc., 2018.
- [15] Jiechuan Jiang, Iñigo Fernández del Amo, and Zongqing Lu. Graph convolutional reinforcement learning for multi-agent cooperation. *CoRR*, abs/1810.09202, 2018.
- [16] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [17] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning.
- [18] Kaixiang Lin, Renyu Zhao, Zhe Xu, and Jiayu Zhou. Efficient large-scale fleet management via multi-agent deep reinforcement learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD ’18*, pages 1774–1783, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5552-0. doi: 10.1145/3219819.3219993.
- [19] Ryan Lowe, YI WU, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6379–6390. Curran Associates, Inc., 2017.
- [20] Nicolas Meuleau, Milos Hauskrecht, Kee-Eung Kim, Leonid Peshkin, Leslie Pack Kaelbling, Thomas Dean, and Craig Boutilier. Solving very large weakly coupled Markov decision processes. pages 165–172, 1998.

- [21] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.
- [22] Santiago Ontanón, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, David Churchill, and Mike Preuss. A survey of real-time strategy game ai research and competition in starcraft. *IEEE Transactions on Computational Intelligence and AI in games*, 5(4):293–311, 2013.
- [23] Scott Proper and Prasad Tadepalli. Solving Multiagent Assignment Markov Decision Processes. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '09, pages 681–688, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-0-9817381-6-1.
- [24] 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. *arXiv preprint arXiv:1803.11485*, 2018.
- [25] Scott Sanner and Craig Boutilier. Practical linear value-approximation techniques for first-order mdps. *arXiv preprint arXiv:1206.6879*, 2012.
- [26] Amanpreet Singh, Tushar Jain, and Sainbayar Sukhbaatar. Individualized controlled continuous communication model for multiagent cooperative and competitive tasks. In *International Conference on Learning Representations*, 2019.
- [27] Satinder P. Singh and David Cohn. How to dynamically merge Markov decision processes. In *Advances in neural information processing systems*, pages 1057–1063, 1998.
- [28] Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, pages 2244–2252, 2016.
- [29] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
- [30] Gerald Tesauro. Online resource allocation using decompositional reinforcement learning. In *AAAI*, volume 5, pages 886–891, 2005.
- [31] Yuandong Tian, Qucheng Gong, Wenling Shang, Yuxin Wu, and C. Lawrence Zitnick. Elf: An extensive, lightweight and flexible research platform for real-time strategy games. *Advances in Neural Information Processing Systems (NIPS)*, 2017.
- [32] Nicolas Usunier, Gabriel Synnaeve, Zeming Lin, and Soumith Chintala. Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks. *arXiv preprint arXiv:1609.02993*, 2016.
- [33] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [34] Pawel Wawrzynski. Control policy with autocorrelated noise in reinforcement learning for robotics. *International Journal of Machine Learning and Computing*, 5(2):91, 2015.
- [35] Yaodong Yang, Rui Luo, Minne Li, Ming Zhou, Weinan Zhang, and Jun Wang. Mean field multi-agent reinforcement learning. *arXiv preprint arXiv:1802.05438*, 2018.
- [36] Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, et al. Relational deep reinforcement learning. *arXiv preprint arXiv:1806.01830*, 2018.

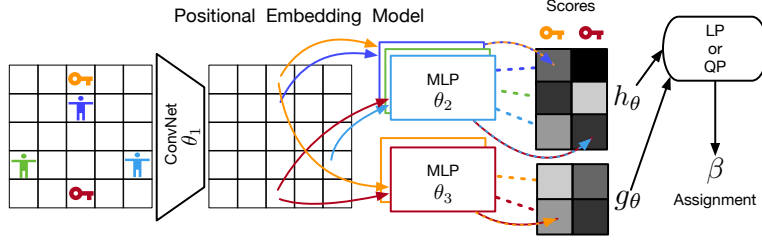


Figure 2: The PEM model for 3 agents and 2 tasks, arrows of interactions are only drawn for 2 agents each with one task, and one interaction of the two tasks. Agents are in cold (blue/green) colors, tasks are in warm (red/orange) colors.

## A The Positional Embedding Model

A more complete illustration of the PEM model is reported in Fig. 2.

---

### Algorithm 1 Structured Prediction RL algorithm - Worker thread

---

```

// p is the number of correlated steps,  $\sigma$  the exploration standard deviation
// We denote by  $Queue_p$  a queue of fixed size p. We assume we can sum all the elements in it.
 $T \leftarrow 0$ 
repeat
  Start new a episode,  $t \leftarrow 0$ 
  for all agent  $i$  and task  $j$  do
    Init noise history for the linear part to 0:  $noise\_hist\_lin(i, j) = Queue_p(0)$ 
  end for
  for all pair of task  $j, k$  do
    Init noise history for the quadratic part to 0:  $noise\_hist\_quad(j, k) = Queue_p(0)$ 
  end for
  repeat
    Observe state  $s_t$ 
    for all agent  $i$  and task  $j$  do
      Compute  $h_\theta(i, j, s_t)$  using the network
      Sample the actual action  $H_{i,j}(s_t) \sim \mathcal{N}(h_\theta(i, j, s_t) + \sum noise\_hist\_lin(i, j), \frac{\sigma}{p})$ 
      Store the current noise  $noise\_hist\_lin(i, j).append(H_{i,j}(s_t) - h_\theta(i, j, s_t))$ 
      Compute  $\mu_{i,j}(s_t)$  the contribution of the agent to the task
      Compute  $u_j(s_t)$  the capacity of the task
    end for
    for all pair of task  $j, k$  do
      Compute  $g_\theta(j, k, s_t)$  using the network
      Sample the actual action  $G_{j,k}(s_t) \sim \mathcal{N}(g_\theta(j, k, s_t) + \sum noise\_hist\_quad(j, k), \frac{\sigma}{p})$ 
      Store the current noise  $noise\_hist\_quad(j, k).append(G_{j,k}(s_t) - g_\theta(j, k, s_t))$ 
    end for
    Compute the assignment using the constrained optimizer  $\beta \leftarrow solve(H, G, \mu, u)$ 
    Execute the assignment in the environment, observe reward  $r_t$ 
    The policy is  $\pi(s_t) = [h_\theta(s_t), g_\theta(s_t)]$ , and the action  $a(s_t) = [H, G]$ 
    Send to the learner thread  $(\pi(s_t), a(s_t), s_t, r_t)$ 
     $t \leftarrow t + 1$ 
     $T \leftarrow T + 1$ 
  until Episode ends
until  $T > T_{max}$ 

```

---

---

**Algorithm 2** Structured Prediction RL algorithm - Learner thread

---

*// p is the number of correlated steps, N is the return length we use,  $\sigma$  the exploration standard deviation*  
 *$\gamma$  is the discount factor,  $\lambda$  is the policy weight*  
**repeat**  
  Reset gradients  $d\theta \leftarrow 0$   
  Receive  $N$  consecutive samples from a worker thread  $[(\pi_1, a_1, s_1, r_1), \dots, (\pi_N, a_N, s_N, r_N)]$   
   $R = \begin{cases} 0 & \text{if } s_N \text{ is terminal} \\ V(s_N, \theta) & \text{otherwise} \end{cases}$   
  **for**  $t = N - 1$  **to** 1 **do**  
     $R = \begin{cases} r_t & \text{if } s_t \text{ is terminal} \\ r_t + \gamma R & \text{otherwise} \end{cases}$   
    Accumulate value loss  $d\theta \leftarrow d\theta + \nabla_{\theta} (|R - V(s_t, \theta)|)$   
    Compute advantage  $A_t \leftarrow R - V(s_t)$   
    Compute new policy  $\pi_{\text{new},t}(s_t, \theta)$  using the network  
    Compute likelihood of action according to old policy (assuming  $a_t \sim \mathcal{N}(\pi_t, \sigma)$ )  $l_{\text{old}} \leftarrow \text{NormalPDF}(a_t, \pi_t, \sigma)$   
    Compute likelihood according to new policy (assuming  $a_t \sim \mathcal{N}(\pi_{\text{new},t}, \sigma)$ )  $l_{\text{new}} \leftarrow \text{NormalPDF}(a_t, \pi_{\text{new},t}, \sigma)$   
    Compute importance ratio  $ir \leftarrow \frac{l_{\text{new}}}{l_{\text{old}}}$   
    Accumulate policy loss  $d\theta \leftarrow d\theta + \lambda \nabla_{\theta} (ir * A * \log(l_{\text{new}}))$   
  **end for**  
  Take an optimization step according to  $d\theta$ .  
**until** training ends

---

## B Formal description of the training algorithm

In the experiments of this paper, we chose to build upon A2C with continuous actions, but in principle, any continuous policy-gradient algorithm could be used. The pseudo-code of the learning algorithm is presented in Algorithm 1 for the worker threads and Algorithm 2 for the learning threads.

## C Search and rescue Experimental Setup

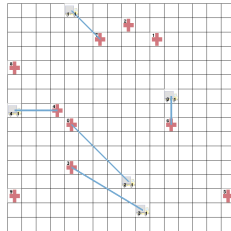


Figure 3: Search and rescue environment, with 5 ambulances and 10 victims.

### C.1 Details of the Problem

We consider a search and rescue problem on a 16x16 grid. Each ambulance can move in any of the 8 adjacent cell at each step, and is assumed to be able to pick-up an infinite number of victims. At the beginning of each episode,  $m$  victims and  $n$  ambulances are spawned uniformly at random on the grid. An ambulance picks up a victim as soon as it reaches its cell, regardless of whether this ambulance was effectively assigned to that victim or if it visited the cell contingently. When it happens, the state of the victim changes to reflect the fact that that task is solved, yet nothing prevents the agent to keep assigning ambulances to it.

## C.2 Training

We optimize the hyper-parameters of the learning procedure using a random search. We sample 128 sets of hyper-parameters for each combination of model/scenario, and we train the models during 8 hours on a Nvidia Volta. We report the performance of the best performing model after training according to the average reward on training problems. The parameters that were tuned are the following: learning rate of the value network (we sample  $c \in [0, 5]$ , uniformly at random, and use  $10^{-c}$ ), the learning rate of the policy network (same sampling), the variance of the random policy (uniformly in  $[0.1, 2]$ ), the number of correlated steps in the exploration (uniformly in  $[1, 10]$ ), the number of steps of the  $n$ -step return (uniformly in  $[2, 5]$ ), and the optimization algorithm (SGD or Adam [16]).

## C.3 Optimal algorithm

For the small instances of the problem, it is possible to use an exact algorithm (topline) that solves the Multi-Vehicle Routing Problem exactly. To do it efficiently, we first pre-compute for each subset of victims (there are  $2^m$  possible subsets), the shortest path to visit them all, starting from each of the victims of the subset (that is, if a subset contains 5 victims, we compute the 5 shortest paths that go through all of them starting from each of the victims). This pre-computation phase is done using a dynamic programming algorithm that has a complexity of  $O(2^m m^3)$ . This allows us to compute, for a given ambulance, what is the optimal way of visiting a given subset of the victims: we simply need to choose which victim to visit first and then execute the optimal path that we have cached between the remaining ones. What remains to be done is to partition the victims in  $n$  sets, and assign those sets to the  $n$  ambulances, so that the maximal time required by the ambulances to visit their subset optimally is minimized. To solve this partition/assignment problem efficiently, we model it as an Integer Linear Program, and solve it exactly using an efficient branch-and-cut solver (SCIP [10])

## C.4 Model

**Features.** The input of the networks consists in 4 feature planes: the first one contains a 1 if the corresponding cell contains a victim (0 otherwise), the second contains a 1 if the corresponding cell contains an ambulance, and the last two planes contain respectively the x/y coordinates of the entity (victim or ambulance) contained in that cell, if there is any.

**Network.** The value network is a residual network ([13]) made of 3 residual blocks of 2 convolutional layers, with kernel size 3, stride 1, and padding such that the output of each layer has the same spatial dimension as the input. Each convolutional layer outputs 32 feature planes, and we use ReLUs as non-linearities, as well as BatchNorm layers for regularization. The output of the last block is connected to a fully connected layer, which outputs one single value, the value function. For the direct model, we use a simple fully connected network, with 3 linear layers separated by ReLUs, with 32 hidden units each. The architecture of the PEM network is the same as the value network.

We train the model using a synchronous Advantage-Actor-Critic algorithm (A2C) (see [21] for the asynchronous version), using the ELF platform [31]. We batch 128 observations together, and run 256 agents in parallel.

## C.5 Detailed results analysis

For the AMAX strategy, the experiments show that the model that has access to the the full view of the map (PEM) performs better on the domain where it was trained, w.r.t. a model that only has access to the features of the objects (DM). This means that the model learns something that is more useful (as a parameterization of the LP /QUAD ) than simply the distance between objects, and probably takes into account some local patterns of the other objects on the map. However, this better performance in-domain comes at a cost of poor generalization: on the  $8 \times 15$  scenario, the AMAX PEM model and the QUAD PEM model trained on the  $2 \times 4$  gets stuck in a loop in a fraction of the evaluation episodes, for which they are not able to complete the overall problem (in the table we report the performance averaged over non-failing episodes). This is probably because the receptive fields of the network are significantly more saturated (their average activations have a higher value) in the  $8 \times 15$  scenario than the models learned in the  $2 \times 4$  training scenario. By contrast, the models trained on  $5 \times 10$  is not affected by this. The phenomenon also occurs in the opposite direction: the QUAD PEM

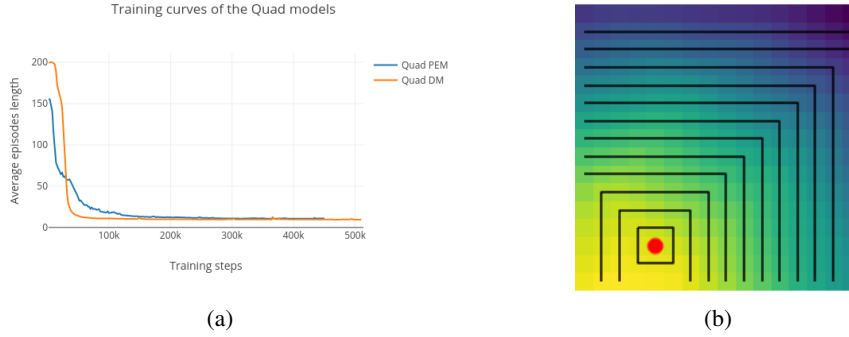


Figure 4: (left) Learning curves of the Quad models; (right) The distance computed by model (depicted as the density), is skewed towards the corner, compared to the ground-truth distance (depicted as the contours)

trained on  $5 \times 10$  experiences failures on the smaller environment  $2 \times 4$ , indicating some form of overfitting.

Another interesting observation is that the AMAX PEM agent performs on average 7% better than the baseline. This is unexpected since the only features it has access to are the positions of the ambulance and the victim, thus it would be expected to learn to send each ambulance to the closest victim. We credit this development to the ability of the model to learn to break ties between equally close victims, which it does by choosing the victims that are on the furthest peripheral positions of the map. Since these positions tend to be the outliers, they are on average more difficult to access than the ones in the middle, so it is beneficial to favor them. The baseline breaks ties randomly and thus does not account for that. This can be observed in Fig 4b in which we plotted the similarity function learned by the model, by computing the score the model would give when the agent is located at the red dot and the task is located anywhere else. While the function it generates resembles the ground-truth distance, the distribution is slightly skewed towards the bottom-left corner.

Overall, the constrained strategies (LP and QUAD) perform better than the AMAX strategy, and the QUAD strategy is better than the LP strategy, irrespective of the feature model used. This is an evidence that the structure introduced into the inference procedure is indeed able to improve the performance of the agent. These strategies also leverage the structure of the problem through the constraints on the assignment, which means that there is less room for learning something that cannot be computed directly from the pairwise positions using the additional information provided by the access to the full view. Since the extra information does not directly help, it slows down the learning procedure, and hinders the final performance. This can be seen in Fig 4a, where the PEM agent is slower to converge, and converges to a worse performance than the DM agent, when both use the QUAD strategy.

## D StarCraft experimental setup

All the experiments are played on a fully empty map, where the units are centered in the middle, outside their respective fire range. Because of the kind of movement policy we use, the units never go near the edge of the map, hence there is no collision involved but the collisions between units themselves. The bot takes its actions every 3 frames, but we reevaluate the target assignment only every 6 frames. If the target of a unit dies in the meantime, then it does nothing until the next re-assignment. The reward at time  $t$  is defined as

$$r_t = (\text{ourHealth}_t - \text{ourHealth}_{t-1} + \text{theirHealth}_{t-1} - \text{theirHealth}_t) / \text{ourHealth}_0.$$

At the beginning of every battle, the units are spawned according to a seeded normal distribution, which is skewed towards the Y dimension, in order to form more coherent initial spread of the units. Even though the starting positions are deterministic with respect to the seed, the outcome of a battle is not, even if the policy of the players are totally deterministic. This is due to the internal randomness of the game engine, which affects things like the random miss probability of each attack (any attack has a  $\frac{1}{256}$  probability of failing) and the initial orientation of the units.

The opponent army is controlled by the built-in AI. In practice, it means that we give an "attack-move" order to the opponent, with the target position being the centroid of our units. This order is repeated every 60 frames with an updated centroid. The attack-move order causes the built-in AI to take over the unit, globally moving it towards the target position and attacking any visible unit along the way.

We wish to emphasize the fact that all the details of the experimental protocol have a significant impact on the outcome of the battles. In particular, the frequency at which the opponent's attack-move command is re-issued matters: if the frequency is too high, then the units tend to attack less, because spamming orders have some weird effects on the game engine. Conversely, if the frequency is too low, then the attack point might be significantly off the centroid of our army, leading to sub-optimal attacking behaviour. Similarly, the initial positions of the units plays an important role. As a result, the exact win-rate are not directly comparable with previous works, where neither precise details of the setup nor source code are available. The differences can be observed even in the win-rate of the baselines heuristics. For this work, we refer to the source code in the supplementary material for the exact details used.

## D.1 Scenarios

We consider three different kinds of scenarios:

**Wrath** These are ranged flying units, which means that they don't collide with any other unit. We denote these scenarios as  $wNvM$  where  $N$  is the size of our army, and  $M$  is the size of the enemy army. Following the previous works, we train on the imbalanced scenario  $w15v17$ , where the two additional units given to the opponent are required to make the scenario challenging.

**Marines** These are ranged ground units, which do have collisions. We denote these scenarios as  $mNvM$  where  $N$  is the size of our army, and  $M$  is the size of the enemy army. Since the built-in AI has a better control policy for ground units, we train on the balanced scenario  $m10v10$ , which is challenging enough.

**Zergling-Hydralisk** Zerglings are fast melee units (they can only attack if they are in contact of their target), while Hydralisks are slower ranged units. In this scenario, we investigate the opportunity to learn distinct behaviours depending on the type of the agent or its task. To further amplify their relative capabilities, we give the Zerglings a speed boost ("Metabolic Boost" upgrade), and the Hydralisks a range boost ("Grooved Spines" upgrade). We denote these scenarios as " $zhNvM$ " which correspond to  $N$  Zerglings and  $N$  Hydralisks versus  $M$  Zerglings and  $M$  Hydralisks. We train on  $zh10v10$ .

## D.2 Baseline Heuristics

Here is a description of the heuristics used as a comparison:

**Closest (c)** Each unit independently picks the units that is the closest, as measured by the distance function used by the game engine. Ties are broken using the unit internal ID, which is randomly assigned at the beginning of the game but consistent for the duration of the episode.

**Weakest Closest (we)** All units are collectively assigned to the weakest enemy unit. The distance is used as a tie breaker.

**Weakest closest No-Overkill (wcnok)** . We select the weakest-closest enemy unit, then assign greedily in an arbitrary order as many units as possible as long as the total sum of damage to this unit is lesser than its health. When this enemy is saturated, if some of our units don't have a target yet, then we select the second weakest-closest, and so on until all enemy units have been exhausted or all our units have a target.

**Weakest Closest No-Overkill No Change(wcnoknc)** Same as  $wcnok$ , but once a unit starts attacking a target, it keeps doing so until the target dies. When the target dies, a new target is computed as in  $wcnok$  Keeping the same target can reduce some instability in the assignment found by  $wcnok$ , but can lead to over-killing.



Experiment	Model	Learn. rate	Policy-loss weight $\lambda$	Explor. std-dev $\sigma$	Returns length	# correlated steps
Marine	QUAD	4.272e-05	1.585e+01	2.910e+00	5	10
	LP	2.280e-05	2.157e-03	7.017e-01	6	5
	AMAX	4.330e-05	5.396e-01	2.418e+00	2	2
Wraith	QUAD	2.826e-05	2.514e+02	1.899e+00	6	7
	LP	5.600e-05	5.534e-01	4.244e-01	4	6
	AMAX	2.525e-05	6.847e+01	1.994e+00	8	10
Zergling-Hydra	QUAD	5.325e-05	6.277e-01	7.185e-01	3	3
	LP	6.534e-05	4.455e-02	2.902e+00	8	1
	AMAX	1.885e-05	2.119e-02	1.881e+00	3	10

Table 3: Best hyper-parameters found through random search on the different models

**Weakest Closest No-Overkill Smart (wcnoks)** Same as wcnoknc, except that the target is kept only as long as it doesn’t risk causing over-killing. When it does, a new target is computed as in wcnok.

**Random No change (rand-nc)** Each unit pick a target at random at the beginning of the episode, and keep attacking it until it is dead. When that happens, a new target is picked randomly.

### D.3 Model

The scoring models  $h_\theta$  and  $g_\theta$  are fully-connected networks consisting of 3 linear layers with ReLU. To compute  $h_\theta(i, j)$ , we give as input to the network the concatenation of the features of ally unit  $i$  and the features of enemy unit  $j$ , along with 2 additional features: a boolean flag that indicates whether  $i$  was attacking  $j$  in the previous step (this is meant to facilitate temporal consistency of the actions), and the distance between both units, as computed by the internal game engine. The input of  $g_\theta(j, k)$  only contains the features of enemy units  $j$  and  $k$ , with no additional features. In this experiment, we use 64 agents in parallel during training, and batch 32 observations together. The reward is joint, and consist in the normalized instantaneous delta of health of our units and the enemy units.

### D.4 Hyper-parameters

All models use the same network architecture, using 3 layers of fully-connected with 32 units and ReLUs in-between. The value function uses a residual network made of 5 blocks of 2 convolutional layers, with 16 feature layers and kernel size 3, operating on a 100x100 view of the state. The discount factor  $\gamma$  is set to 0.999.

For all the models, we sample randomly the remaining hyper-parameters, as follows: For the learning rate we sample  $c$  u.a.r in  $[-6, -3]$ , and use  $10^c$ , for the policy-loss weight  $\lambda$ , we sample  $d$  u.a.r in  $[-3, 3]$  and use  $10^d$ , the exploration standard deviation  $\sigma$  is sampled u.a.r in  $[0.1, 3]$ , the return-length used in A2C is an integer sampled u.a.r in  $[2, 10]$ , and the number of correlated exploration steps is an integer sampled u.a.r. in  $[1, 10]$ . We report the best parameters found for all model in all experiments in Table 3

### D.5 Detailed Results

In Table 4, we report the performance of all the heuristics, as well as the confidence intervals. In the Marine and Zergling-Hydralisk scenarios, the “closest” heuristic tends to dominate, suggesting that there is not much value in learning a focus-firing policy. However, in the Wraith scenarios, the best performing heuristic are the more complicated ones that focus fire on the weakest while preventing over-killing. This is the type of policies that only the QUAD model is able to represent, explaining its dominance in this scenario. AMAX and LP manage to do better than the closest heuristic by using enemy positions as a way to reach a consensus: they tend to favor picking targets which have a high  $y$  coordinate, for example, so that they all end-up picking the same in the beginning, creating ad-hock collaboration. However, as the battle goes on, this coordination scheme is not as efficient, since the battle tend to be messy.

Train	Test	Heuristics						RL		
		c	wc	wenok	wcnoknc	wcnoks	rand-nc	LP DM	QUAD DM	AMAX
m10v10	m5v5	<i>0.77 ± 0.03</i>	<i>0.88 ± 0.02</i>	0.56 ± 0.03	0.86 ± 0.02	0.83 ± 0.02	0.15 ± 0.02	<b>0.90 ± 0.02</b>	0.83 ± 0.02	0.84 ± 0.02
	m10v10	<i>0.77 ± 0.03</i>	0.44 ± 0.03	0.00 ± 0.00	0.45 ± 0.03	0.56 ± 0.03	0.01 ± 0.01	<b>0.94 ± 0.02</b>	0.83 ± 0.02	0.82 ± 0.02
	m10v11	<i>0.25 ± 0.03</i>	0.07 ± 0.02	0.00 ± 0.00	0.05 ± 0.01	0.11 ± 0.02	0.00 ± 0.00	<b>0.52 ± 0.03</b>	0.28 ± 0.03	0.29 ± 0.03
	m15v15	<i>0.75 ± 0.03</i>	0.03 ± 0.01	0.00 ± 0.00	0.14 ± 0.02	0.18 ± 0.02	0.00 ± 0.00	<b>0.92 ± 0.02</b>	0.69 ± 0.03	0.77 ± 0.03
	m15v16	<i>0.40 ± 0.03</i>	0.00 ± 0.00	0.00 ± 0.00	0.03 ± 0.01	0.02 ± 0.01	0.00 ± 0.00	<b>0.68 ± 0.03</b>	0.32 ± 0.03	0.43 ± 0.03
	m30v30	<i>0.69 ± 0.03</i>	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	<b>0.74 ± 0.03</b>	0.06 ± 0.02	0.36 ± 0.03
w15v17	w15v17	<i>0.07 ± 0.02</i>	0.00 ± 0.00	0.58 ± 0.03	0.33 ± 0.03	<i>0.81 ± 0.02</i>	0.01 ± 0.01	0.53 ± 0.03	<b>0.89 ± 0.02</b>	0.30 ± 0.03
	w30v34	0.01 ± 0.01	0.00 ± 0.00	0.36 ± 0.03	0.31 ± 0.03	<i>0.90 ± 0.02</i>	0.01 ± 0.01	0.76 ± 0.03	<b>0.99 ± 0.01</b>	0.37 ± 0.03
	w30v35	0.00 ± 0.00	0.00 ± 0.00	0.10 ± 0.02	0.08 ± 0.02	<i>0.60 ± 0.03</i>	0.01 ± 0.00	0.56 ± 0.03	<b>0.94 ± 0.02</b>	0.24 ± 0.03
	w60v67	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	<i>0.07 ± 0.02</i>	0.00 ± 0.00	0.33 ± 0.03	<b>0.72 ± 0.03</b>	0.13 ± 0.02
	w60v68	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	<i>0.01 ± 0.01</i>	0.00 ± 0.00	0.21 ± 0.03	<b>0.52 ± 0.03</b>	0.07 ± 0.02
	w80v82	0.00 ± 0.00	0.00 ± 0.00	0.09 ± 0.02	0.08 ± 0.02	<i>0.32 ± 0.03</i>	0.00 ± 0.00	0.11 ± 0.02	<b>0.36 ± 0.03</b>	0.03 ± 0.01
zh10v10	zh10v10	<i>0.86 ± 0.02</i>	0.26 ± 0.03	0.00 ± 0.00	0.54 ± 0.03	0.64 ± 0.03	0.00 ± 0.00	<b>0.90 ± 0.02</b>	0.83 ± 0.02	0.84 ± 0.02
	zh10v11	<i>0.30 ± 0.03</i>	0.01 ± 0.01	0.00 ± 0.00	0.04 ± 0.01	0.09 ± 0.02	0.00 ± 0.00	<b>0.46 ± 0.03</b>	0.24 ± 0.03	0.40 ± 0.03
	zh10v12	<i>0.03 ± 0.01</i>	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	<b>0.06 ± 0.02</b>	0.01 ± 0.01	<b>0.06 ± 0.01</b>
	zh11v11	<b>0.87 ± 0.02</b>	0.15 ± 0.02	0.00 ± 0.00	0.38 ± 0.03	0.56 ± 0.03	0.00 ± 0.00	<b>0.87 ± 0.02</b>	0.75 ± 0.03	0.80 ± 0.02
	zh12v12	<b>0.85 ± 0.02</b>	0.05 ± 0.01	0.00 ± 0.00	0.23 ± 0.03	0.42 ± 0.03	0.00 ± 0.00	0.82 ± 0.02	0.64 ± 0.03	0.75 ± 0.03

Table 4: Results on StarCraft<sup>®</sup>: average win-rate of the different methods and all the heuristics. Bests results are in bold, the best heuristic on each scenario is in italics. We report 95% confidence intervals (using the Normal approximation interval).

In the Marine scenario, all three models usually learn to wait in the beginning of the battle, and start picking a target only when the enemies are about to get in range. This strategy allows them to keep a rather good formation, which they would otherwise lose, had they picked a wrong target. This gives them a little edge in the battle, and then they proceed by following a strategy that resembles attacking the closest unit.

In the Zergling-Hydralisk scenarios, the models learn to wait in the beginning as well. This is an exploit on the rushing behaviour of the opponent: the faster zerglings of the enemy will engage first, while its hydralisks are lagging behind. This allows the models to first clear up the zerglings wave with the combined forces of their zerglings and hydras, before turning to the enemy hydralisks.

We provide some replay video as a supplementary file.