The Performance Impact of Combining Agent Factorization with Different Learning Algorithms for Multiagent Coordination

Andreas Kallinteris  
Technical University Of Crete  
Chania, Greece  
akallinteris@isc.tuc.gr

Stavros Orfanoudakis  
Technical University Of Crete  
Chania, Greece  
sorfanoudakis@isc.tuc.gr

Georgios Chalkiadakis  
Technical University Of Crete  
Chania, Greece  
gehalk@intelligence.tuc.gr

ABSTRACT
Factorizing a multiagent system refers to partitioning the state-action space to individual agents and defining the interactions between those agents. This so-called agent factorization is of much importance in real-world industrial settings, and is a process that can have significant performance implications. In this work, we explore if the performance impact of agent factorization is different when using different learning algorithms in multiagent coordination settings. We evaluated six different agent factorization instances—or agent definitions—in the warehouse traffic management domain, comparing the performance of (mainly) two learning algorithms suitable for learning coordinated multiagent policies: the Evolutionary Strategies (ES), and a genetic algorithm (CCEA) previously used in this setting. Our results demonstrate that different learning algorithms are affected in different ways by alternative agent definitions. Given this, we can deduce that many important multiagent coordination problems can potentially be solved by an appropriate agent factorization in conjunction with an appropriate choice of a learning algorithm. Moreover, our work shows that ES is an effective learning algorithm for the warehouse traffic management domain; while, interestingly, celebrated policy gradient methods do not fare well in this complex real-world problem setting.

1 INTRODUCTION
Most multiagent systems (MAS) benchmarks and frameworks assume that the structure of the system and the concept of the agent are predetermined, as in the well-known prisoner’s dilemma and predator-prey problems. In many modern industry-oriented domains, however, it is imperative that the system designer is allowed to factorize the agent’s definition for the problem, according to her requirements [9]. Such domains include network routing [11, 26], autonomous warehouse traffic management [7, 10], and power plant control [23]. For example, in an autonomous warehouse traffic management domain, a MAS architect could design a system such that every autonomous ground vehicle (AGV) is controlled by a single agent; or they could design the system such that every corridor of the warehouse is controlled by a single agent; or they could give a single agent control of the entire warehouse. In general, whenever the domain enables factorizable agent definitions, there exist many ways to break down the problem into a multiagent system, which may differ on what state each agent can observe, which actions it can take, and how the agents interact with each other.

There has been some research on the effects of changing the state resolution of an agent for more efficient training while keeping the (multi-) agent structure constant [18]. However, even though most systems enable factorizable agent definitions, there is very little to no research on the impact of different agent definitions on the performance of the system—with the exception of [6], which compares six different agent definitions using the same learning algorithm in a warehouse traffic congestion management domain.

Moreover, to the best of our knowledge there has been no research on the impact of agent definitions in conjunction with using different learning algorithms in the same domain. Analyzing the impact of changing agent definitions using multiple learning algorithms is both challenging and important: it is challenging, because factorizing the problem in a specific way is likely to make the problem a lot harder to solve for some learning algorithms, and as such there may not be multiple agent definitions (multiagent team factorization) suitable for multiple learning algorithms. To put it differently, in order for a specific multiagent (MA) learning algorithm to be able to operate properly in a complex domain, it is usually required that a specific agent definition corresponding to a small set of possible MAS structures is used; and this set may have no overlap with a set of possible MAS structures required by a different learning algorithm. At the same time, evaluating alternative agent definitions in conjunction with different learning algorithms is important, as it can (a) identify appropriate such pairings to solve specific complex problems effectively; (b) also provide key intuitions on generic attributes such pairings must possess in order to solve more general classes of MA learning and
coordination problems; as well as (c) provide further insights on the inner properties of the learning algorithms that make them good candidates for particular domains.

Against this background, in this work we explore the performance impact of agent factorization in conjunction with using different learning algorithms in multiagent coordination settings. We showcase this impact while focusing on the trending multiagent warehouse traffic management problem [5]. This problem is particularly challenging for traditional reinforcement learning (RL) algorithms to solve, due to its high dimensional state space, continuous actions space, and sparse reward signals. We tested various learning algorithms in this domain, including the cooperative co-evolutionary genetic algorithm (CCEA) [21], which previous research has shown to be successful in this problem domain [7]; and the more elaborate Evolutionary Strategies (ES) optimization algorithm [24] that is known to constitute a scalable alternative to RL methods in high dimensional spaces with sparse rewards, while also extending it to work for MA systems (MA-ES). Our primary findings and contributions are the following:

- The learning performance impact of different agent definitions, differs based on the learning algorithm used.
- The use of different agent definitions has a greater impact on more complex environments, e.g. when the number of training parameters significantly increases.
- We provide intuitions on the pairings of agent factorization with learning algorithms, discussing how these are successful in or incapable of tackling the problem at hand.

While, our secondary contributions are the following:

- Celebrated MARL algorithms, like Independent DQN [17], and MADDPG [15], are shown to be unable to solve the warehouse traffic management problem.
- By contrast, evolutionary approaches fare much better in this complex domain that necessitates delayed rewards and poses a credibility assignment problem.
- In particular, (MA-)ES is an effective learning algorithm for the warehouse traffic management domain.

2 BACKGROUND AND RELATED WORK

Agent factorization is the process of breaking down the domain’s state-action space into subsets to be controlled by each agent, and defining the communications between the agents [9]. Appropriately factorized MA architectures are required for domains where naturally occurring entities can not be easily modeled into individual agents. For example, in a complex computer network routing domain, an appropriate (potentially hierarchical) factorization of the problem is required in order to achieve reasonable performance. When factorizing a system, it is important that each agent controls a subset that logically corresponds to a set of connected entities of the environment. To give a “negative” example, in robot soccer a MA architecture with two agents, where one agent controls the left leg of each robot player and the other controls the right leg of each robot player, is an architecture where each agent controls a set of not directly associated entities of the environment.

A recent work on the impact of agent factorization is [6], where the researchers compare six different agent definitions using the same learning algorithm (CCEA) on a warehouse traffic congestion management domain with various complexity levels, showing that agent definition has a significant performance impact on the system. There, the term “agent definition” refers to the way the state-action information is split to create new agent teams.

The warehouse traffic management is indeed an interesting optimization problem, attracting much work regarding how autonomous ground vehicles (AGVs) should operate [8, 22]. Regardless, the authors of [7] showed that even a simple genetic algorithm, CCEA, can adequately solve the warehouse traffic management problem—at least in low complexity environments involving up to 120 AGVs.

Indeed, genetic algorithms are used to solve a wide variety of optimization problems notwithstanding the special domain traits, because of their unique “Darwinian” approach [13]. The main characteristics of these algorithms are the mutation and crossover steps, which, along with a carefully chosen fitness function, provide a quite efficient action exploration rate. The CCEA used in [7] is a multi-agent genetic algorithm originally introduced by [21] for general optimization problems with many functions.

2.1 Warehouse Traffic Management

Warehouses are essential in every logistic system in the world [16]. Nowadays, more and more warehouses require fewer human workers, especially in inter-warehouse logistics, because of the breakthrough of autonomous ground vehicles (AGVs) [25]. AGVs require effective coordination algorithms in order to operate efficiently, and traditional optimization techniques are not sufficient to properly route AGVs to their destination without avoiding congestion [1].

In this work, we explore how different learning algorithms for AGV coordination perform when paired with six different agent definitions, in the warehouse traffic management domain introduced in [7]. There, low AGV count represents simple domains, while higher AGV count corresponds to more complex ones.

The Warehouse, shown in Figure 1, is represented by a Directed Graph $G = (V, E)$, where the edges $e \in E$ represent the corridors of the warehouse and are characterized by their AGV capacity $c_\text{ape} \geq \frac{n_e}{n_e}(t)$ representing its physical area in a warehouse, and AGV traverse time $c_\text{travel}^e$, and the vertices $v \in V$ represent stock points which do not have a capacity— instead, when an AGV is waiting for a congestion to be resolved, it counts as being in the edge it was coming from. Each AGV plans its path using $A^*$ search on the traversal cost of the graph. The cost of traversing an edge is:

$$\text{cost}_e^{\text{total}} = \text{cost}_e^{\text{travel}} + \text{cost}_e^{\text{add}}(t)$$

where $\text{cost}_e^{\text{add}}$ is an additional cost added by the MA congestion avoidance system. At the start of the simulation the AGVs are positioned at various entry points across the graph, and when a delivery is completed the AGV is immediately assigned a new mission to accomplish starting from its current location, resulting in a constant number of AGVs.

Here, the main challenge for this domain is to handle the AGVs’ traffic in a way that maximizes the total throughput, while minimizing the traffic congestion. To this end, the traffic management system has to learn how to find the most suitable additional costs $\text{cost}_e^{\text{add}}(t) \forall e \in E$ given the current state $S$ of the warehouse, where...
Can wait at a vertex until an edge is empty. While in this queue, it continues to count towards the capacity of its current edge.

Thus, instead of using optimization techniques, a team of agents can be used, where each agent observes and controls a subset of the set of edges that it controls. This problem cannot be practically solved with traditional RL optimization techniques due to its high dimensionality among other parameters, like learning algorithm used and AGVs count.

In this section, we describe six agent definitions, first introduced in [6], which vary in:

1. **Agent Structure**: the way the graph is partitioned into subsets (coarser vs finer), resulting in a different number of agents, introducing a trade-off between the dimensionality of each agent, and the dimensionality of the coordination problem.
2. **State resolution**: changing what portion of their domain the agents can observe, introducing a trade-off of high dimensionality vs partial observability.

**Centralized Agent.** The first agent variant we investigate is the simplest conceptually, because only a single agent is assigned to manage all the edges of the graph $|I| = 1$. The centralized agent’s state $s$ is defined as the total number of AGVs currently traversing each edge $n_e(t)$, while the centralized agent’s action is the additional cost of travel for every edge $\text{cost}_{add}$.

\[
\begin{align*}
    s_{\text{Central}}(t) &= [n_e(t)] & \forall e \in E, \\
    a_{\text{Central}}(t) &= [\text{cost}_{add}(t)] & \forall e \in E.
\end{align*}
\] (3)

From the agent’s perspective, this is a very complex, high dimensional learning problem, however, from the team’s perspective, it is the simplest since there is no other agent to cooperate with.

**Link Agent.** The second variant we investigate is the exact opposite of the centralized agent. Here, every link agent $i$ is responsible for a single directed edge $e_i$, so the MA team consists of $|I| = |E|$ agents. The link agent’s state $s_i$ is the total number of AGVs which traverse the edge $e_i$ at a given time $n_{e_i}(t)$, while the link agent’s action is the additional cost of travel for that edge $\text{cost}_{add}^{e_i}$.

\[
\begin{align*}
    s_{i,\text{Link}}(t) &= n_{e_i}(t), \\
    a_{i,\text{Link}}(t) &= \text{cost}_{add}^{e_i}(t). \\
\end{align*}
\] (4)

This agent definition creates the most decentralized teams, resulting in the most challenging MA team coordination setup. Because it is the most decentralized definitions, it is the most affected by the structural credit assignment problem [1], specifically the agents receive the total team performance as a learning signal, without an indication of individual contributions. In larger warehouses where there are more agents, contribution becomes more ambiguous and noisy. However, each individual agent is very simple, since $|s_{i,\text{Link}}| = |a_{i,\text{Link}}| = 1, \forall i \in I$ making its training straightforward.
Intersection Agent. The third variant we investigate is a middle ground between the completely centralized and completely decentralized link team. Here, each of these agents is responsible to control the AGV traffic on the set of all incoming edges $e_i$ of a vertex $v$ of the graph $G$. So, the team consists of $|L| = |V|$ intersection agents, whose states and actions are:

$$s^\text{Intersection}_i(t) = [n_e(t)] \quad \forall e \in e_i,$$

$$a^\text{Intersection}_i(t) = [\text{cost}_e^{\text{add}}(t)] \quad \forall e \in e_i,
(5)$$

The state-action space for each intersection agent is $|s^\text{Intersection}_i| = |e_i|$ which is very likely to create heterogeneous agents, since not all vertices have the same number of incoming edges. Furthermore, the number of intersection agents is lesser than that of Link and is much greater than that of the Centralized agent, as we generally expect a warehouse graph to have $1 < |V| < |E|$. Therefore, it is also a middle ground with regard to the challenges of structural credit assignment, and the dimension size of the learning problem for each individual agent.

Incorporating Travel Time. All the previous variants focused on the effect of team structure on the MA learning problem without altering the state resolution, where the state information only consists of the current number of AGVs traversing on the edges $n_e(t)$. The variants incorporating travel time investigate the effect of including additional AGV tracking information. The observable state of each edge is augmented by $d_e(t)$ which represents the amount of time remaining until the next AGV completes its traversal. $d_e(t)$’s value can range from the total time required to traverse an edge down to 1, or when the edge is empty 0.

$$s^\text{Central,Time}_i(t) = [n_e(t), d_e(t)] \quad \forall e \in e_i.$$

The fitness score of every gene determines the next steps of mutation and crossover. Typically, only the k best genes are selected to crossover and mutate while the other ones are discarded. This process is repeated until a certain point of optimization is reached.

Cooperative Coevolutionary Algorithm. CCEA [21] is a multigenetic algorithm with its main goal being to coordinate the actions of a set of agents so that they collectively solve a joint problem. The approach uses a genetic algorithm at its core, but with the fitness function reflecting the total utility of all coordinating agents at the end of a simulation; i.e., the fitness function has to be a joint fitness function modeling the coordination problem.

Algorithm 2: CCEA

1. for each agent $a$ do
2. \hspace{1em} $\text{pop}^a_0 = \text{new randomly initialized population}$
3. for each iteration $t = 0, 1, 2, \ldots$ do
4. \hspace{1em} for each agent $a$ do
5. \hspace{2em} Evaluate Fitness of the genes in $\text{pop}^a(t)$
6. \hspace{2em} Select $k$ genes with the highest Fitness value
7. \hspace{2em} $\text{pop}^a_{(t+1)} = \text{Mutate and Crossover the } k \text{ genes}$

As seen in Algorithm 2, the core of the approach is that of a Genetic Algorithm. Therefore, in a similar manner as the Genetic Algorithm, each gene of every agent is evaluated based on its fitness function. Specifically, in line 6 we can see that the selection step is based on the fitness of the genes, then in line 7 the selected ‘fittest’ genes of every agent are crossed over and mutated so that a new population of $N$ genes will be created.

Evolutionary Strategies. Evolutionary strategies [4] is a class of optimization algorithms that has many similarities with the typical genetic algorithms. For example, these methods depend on a huge population of children with different characteristics each, although what mainly differentiates them, is the way of creating the next generations. In this paper, we will focus on a modern variation of Evolutionary Strategies [24]. This algorithm does not select only the genes with the highest fitness score to create the next generations, but it combines the information of every gene.

Algorithm 3: Single-Agent Evolutionary Strategies

Data: Learning rate $\alpha$, noise standard deviation $\sigma$

1. $\theta_0 = \text{randomly initialized policy}$
2. for each iteration $t = 0, 1, 2, \ldots$ do
3. \hspace{1em} Sample $\phi_1, \ldots, \phi_n \sim N(0,1)$
4. \hspace{1em} $F_j = F(\theta_t + \sigma \phi_j) \forall \text{samples } j \in [1,n]$
5. \hspace{1em} Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n \sigma} \sum_{j=1}^n F_j \phi_j$

In detail, Algorithm 1 depicts the original genetic algorithm. Initially in line 1 a population of $N$ randomly initialized genes is created. Genes could be one single parameter or a set of parameters of the system that we want to optimize. Each gene can be evaluated using the fitness function which is a utility function specified by the designer, and is usually based on domain specific properties. The fitness score of every gene determines the next steps of mutation and crossover. Typically, only the 5 best genes are selected to crossover and mutate while the other ones are discarded. This process is repeated until a certain point of optimization is reached.

### 2.3 Learning Algorithms

In this section we present the algorithms used in this work.

Genetic Algorithms [12]. These are a class of optimization algorithms that are famous for their ability to optimize many difficult real life tasks from the simple shift assignment problem to more complex problems of optimization of warehouse traffic. The base idea is that an initially random population of genes will repeatedly mutate and crossover using the fittest genes, until the the gene with the highest fitness score is discovered.
Algorithm 3 shows how this Evolutionary Strategies algorithm works. At first, it creates a randomly initialized base policy \( \theta_0 \) as shown in line 1. Then in line 3, for every iteration \( t \) it samples \( n \) values from a random distribution and adds them as noise to each gene, for example, a Gaussian process is suitable for creating such samples. After that, it performs a simulation with each gene and evaluates them using the fitness function. Finally, the new policy \( \theta_{t+1} \) is created by adding to \( \theta_t \) a weighted sum calculated using the fitness score of every sample in the current iteration as described in line 5 of the algorithm. By doing that the policy \( \theta \) slowly converges to a solution with the best fitness.

Independent Deep Q Networks (I-DQN). In (single or multi-agent) reinforcement learning, every action does get rewarded positively or negatively. By doing that, the agents learn how to maximize this Q-Value function by choosing the appropriate actions. Traditional Q-learning algorithms use a table Q(s, a) to store the evaluation for every possible combination of state and action. Deep Q-Networks (DQN) [17], a celebrated method for deep reinforcement learning, brings together Q-learning with neural networks to allow for function approximation. Extending DQN to multiple agents, calls for a neural network for each agent, in order to evaluate the score of the different state and action combinations.

Multi-Agent Deep Deterministic Policy Gradient. MADDPG [15] is a multi-agent policy gradient algorithm where each decentralized agent learns a critic based on the observations and actions of all agents. The critic is trained using information about the policies of other agents, while the actor is trained using the local information only. This means that the agents’ learned policies use only information from agents’ own observations during execution, and can be applied to cooperative and competitive environments alike. In the end, only the local actors are used at the execution phase, acting in a decentralized manner, while the critics are only used for training.

3 LEARNING COORDINATED MULTIAGENT POLICIES

In this section, we describe the main algorithm we developed and used to experiment with the various agent definitions in order to investigate how different learning algorithms affect the training performance. Also, we discuss the experimental parameters we used for this warehouse traffic management domain.

3.1 MA Evolutionary Strategies

As discussed earlier, there are many ways of creating the next generations of genes. In this work we only focused on one that uses simulated stochastic gradient ascent [24]. Specifically, in a learning problem, \( F() \) will be the stochastic fitness function produced directly by the environment, and \( \theta \) will be the stochastic policy of the agent. Having these in mind we can then write the objective to be maximized in terms of policy \( \theta \), and which can be approximated with sampling as:

\[
\nabla_{\theta} E_{\phi \sim N(0,1)} F(\theta + \sigma \phi) = \frac{1}{\sigma} E_{\phi \sim N(0,1)} \{ F(\theta + \sigma \phi) \phi \} \tag{7}
\]

In Equation 7, \( \phi \) is a random sample of a Gaussian noise distribution and \( \sigma \) is the standard deviation. Using the gradient in Equation 7 as the objective described in terms of policy \( \theta \), Algorithm 4 (MA-ES)\(^1\) extends the ES [24] algorithm to multiagent settings in a straightforward manner. Specifically, MA-ES is an extension of ES which instead of evolving the policy of only an agent per epoch, it evolves the policies of many agents.

Algorithm 4: Multi-Agent Evolution Strategies

\[
\textbf{Data:} \text{Learning rate } \alpha, \text{ noise standard deviation } \sigma, \quad I = \{ h, i, \ldots \} \text{ the set of agents} \\
\begin{align*}
1 & \quad \text{for each agent } i \text{ do} \\
2 & \quad \theta_0^i = \text{randomly initialized policy} \\
3 & \quad \text{for } t = 0, 1, 2, \ldots \text{ do} \\
4 & \quad \text{Sample } \phi_1, \ldots, \phi_n \sim N(0,1) \\
5 & \quad F_j = F(\{ \theta_0^1, \theta_1^1, \ldots, \theta_t^{I-1} \} + \sigma \phi_j) \forall \text{ samples } j \in [1,n] \\
6 & \quad \text{for each agent } i \text{ do} \\
7 & \quad \theta_{t+1}^i \leftarrow \theta_t^i + \alpha \frac{1}{n} \sum_{j=1}^{n} F_j \phi_j
\end{align*}
\]

MA-ES consists of many iterations with two phases each. Initially, every agent \( i \) gets a randomly initialized policy \( \theta_0^i \) in line 2. Then, in line 5, the first phase has to do with the stochastic “mutation” of the populations’ policy parameters, as well as evaluating the results of these perturbations after an episode in the environment. In the second phase, it combines the results of every “child”, for every agent in the team, according to how well they performed, creating a gradient estimate, and updating the parameters (line 7).

3.2 Experimental Parameters

The most important parameter of (MA-)ES is the population size or simply the number of samples \( \phi \) for each training iteration, which occurs because every sample is in fact a new simulation but with some slight differences caused by the perturbation. We found out that the higher the population size, the more the training performance is, and also the higher the exploration rate we get. Intuitively, when there are a lot of samples in an iteration the impact of the outliers (local maximum performance genes) decreases, which means that then we have the chance to get better results converging on a higher local maximum or even in a hard to tell global maximum. However, one of the main weaknesses of MA-ES is that it is computationally expensive, thus we cannot have as large population sizes as we might have wanted. Even so, after many simulations, we concluded that the number of 1000 samples per epoch is sufficient for our current optimization problem.

Furthermore, every sample requires a team of agent policies each parameterized by a neural network(NN). In our experiments, every NN has one hidden layer with 64 nodes and an input size equals the state space of the agent \(|a_{agent}|\) while the output size equals the action space of the agent \(|a_{agent}|\) as defined in the previous section about the different agent definitions. In our case, both CCEA and MA-ES use the same fitness function (equation 8), which describes the total number of successful deliveries \( G \) of a single iteration for all the coordinating agents. So, the Fitness function is a joint fitness function for all agents, modeling their coordination problem.

\(\text{MA-ES with a single agent is the same as ES}\)
Fitness(t) = total_deliveries(t) (8)

The mutation and crossover steps vary in different applications, for instance, the simplest way to mutate genes is the addition of noise $\phi_j$ to the agent policy parameters $\theta_i$ as shown in line 5 of Algorithm 4. Finally, we can clearly see the main differences between (MA)-ES and CCEA. In detail, (MA)-ES combines the information attained from all the genes (line 7 of Algorithm 4), while CCEA uses the information from the k-fittest genes only (line 7 of Algorithm 2), which leads to different learning behaviors, as seen next.

4 EXPERIMENTAL EVALUATION

Here we provide a thorough evaluation and analysis of our approach. We used the warehouse design of Figure 3. The same warehouse was initially tested by [7] to study the impact of different agent definitions on CCEA’s learning performance. We adopt the same experimental setup with [7] to study the impact of different learning algorithms in conjunction with different agent definitions.

We run 30 statistical runs for every different scenario (agent definition, number of AGVs, and learning algorithm) in order to account the randomness of the results. Every unique statistical run consisted of 500 training epochs, and each epoch was divided into 200 simulation steps. At the start of each epoch, the location of the AGVs was fixed having them split at the two sides of the warehouse (half of them were positioned at vertex 0 and the rest at vertex 1, as seen in Figure 3). Also, the NNs’ weights were initialized randomly.

4.1 Evaluating (MA-)Evolutionary Strategies

We begin by evaluating the performance of (MA-)ES in this domain. In Figure 4, we can see the average team performance across 500 training epochs. We define as “team performance” the amount of the completed deliveries $G$ by all the AGVs in a 200-step simulation.

In Figure 5 we can observe the best team performances during training. In the simplest case (90 AGVs) we see that all agent definitions are able to reach the maximum possible performance (554 deliveries), but the centralized agents are not able to do so consistently. This happens because the centralized agents are barely able to handle the dimensionality of the problem. In the 120 AGVs case all agent definitions, other than the centralized with time, are able to reach a similar (non-optimal) result, indicating that for the 120 AGV case the centralized with time agent is unable to handle the high dimensionality of the problem. Most importantly, in the 200 AGVs case even though the dimensionality of the problem has increased, the centralized agents are not the worst-performing ones instead the link agent teams are, indicating that the impact of the credit assignment problem increases faster than that of the curse of dimensionality, while the intersection agent teams that do not struggle as much with those issues significantly outperform others.

On the other hand, in the more complex 400 AGVs scenario, we can see that this trend reverses and the link agents are again not the worse performing ones, indicating that when we get to very complex domains the curse of dimensionality will end up being the biggest obstacle to the performance of the system. Meanwhile, the intersection agents are still the best-performing ones, showing that in complex environments the middle ground (intersection agents) is a good solution to achieve the best performance.

Figure 6 shows the distributions of the completed deliveries at the final epoch for every case after 30 statistical runs. In the simple 90 AGVs case, we can see that even though all agent teams managed to achieve the optimal outcome, when the team does not include time information in its observable state (with time), then the more centralized the agent team is, the more consistently the optimal policy was reached. In the other hand, when the time information was added the opposite is true indicating that the impact of state resolution and state partition are interlinked. Another interesting observation is that in the 90 & 120 AGVs scenarios, the more decentralized the MA system is, the higher the normalized variance we get, this is an indicator of higher exploration rates. While as the complexity of the environment increases in the 200 & 400 AGVs scenarios the highest normalized variance is observed in the intersection agents. One final observation is that the more complex the environment the greater the impact by increasing the observable space (with time) of the agents.

4.2 Comparing CCEA with (MA-)ES

Results in the previous section verified that indeed the agents’ definitions can make a difference in some multiagent environments. We now move to compare (MA-)ES with the CCEA algorithm, which has been in the past the method of choice for tackling this problem domain [6, 7].

Table 1 shows the best performances of both algorithms in all different scenarios. In the 90 AGVs scenario, (MA-)ES managed to...
Converge to the optimal policy with all agent definitions, while CCEA only managed to achieve this for the link with time. In the other scenarios, (MA-)ES achieved a much greater final number of completed deliveries compared to that of the CCEA. Most specifically, we can see that in more complex scenarios (those with higher AGVs count), the improvement gap of the (MA-)ES algorithm is much larger than that of simpler domains (lower AGVs count). For example, in the 90 AGVs case, (MA-)ES managed to generate the optimal team agent policy in every case, while CCEA only achieved it with the link with time agent definitions even though it was quite close to it with the other agent definitions. However, in the 200 AGVs scenario, the (MA-)ES-generated policy had 19 more deliveries in the worst case (link-time agents), and 115 more in the best case (link agents). This shows that this simulated stochastic ascent algorithm (MA-ES) works better than a selection-based genetic algorithm (CCEA).

Looking at the impact of state resolution (incorporating time) on “maximum deliveries” performance, we can observe that in all cases it is mostly insignificant, except for the complex cases of 200 AGVs & 400 AGVs link time agents (893 vs 874 for 200 AGVs and 929 vs 899 for 400 AGVs), indicating that the impact of state resolution is most prevalent in complex environments with high decentralization of agent structures. Following this, we can further investigate the differences between the two algorithms by comparing the way they learn through the course of 500 training epochs. Figure 7 depicts the average deliveries for (MA-)ES and CCEA. Figure 8 compares the best teams performances (i.e., in terms of maximum deliveries attained).

One interesting observation is that, for both algorithms, the decentralized agent teams tend to take big exploration steps without thoroughly considering if performance is going to drop: this is

**Table 1:** Comparison between the maximum deliveries \( G \) of Multiagent Evolution Strategies, and the CCEA. Bold numbers correspond to the maximum deliveries attained by each algorithm for every AGV category.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>90 AGVs</th>
<th>120 AGVs</th>
<th>200 AGVs</th>
<th>400 AGVs</th>
</tr>
</thead>
<tbody>
<tr>
<td>centr.</td>
<td>MA-ES</td>
<td>554</td>
<td>538</td>
<td>598</td>
</tr>
<tr>
<td>centr.-time</td>
<td>MA-ES</td>
<td>554</td>
<td>525</td>
<td>693</td>
</tr>
<tr>
<td>inter.</td>
<td>MA-ES</td>
<td>554</td>
<td>519</td>
<td>695</td>
</tr>
<tr>
<td>inter.-time</td>
<td>MA-ES</td>
<td>554</td>
<td>549</td>
<td>700</td>
</tr>
<tr>
<td>link</td>
<td>MA-ES</td>
<td>554</td>
<td>544</td>
<td>696</td>
</tr>
<tr>
<td>link-time</td>
<td>MA-ES</td>
<td>554</td>
<td>544</td>
<td>696</td>
</tr>
</tbody>
</table>

**Figure 4:** Average team performance across 500 training epochs after 30 statistical runs, using the ES or MA-ES learning algorithms. The optimal solution is different in every different scenario and is directly affected by the different AGV number.

**Figure 5:** Best team performance across 500 training epochs after 30 statistical runs, using the ES or MA-ES learning algorithms.
depicted by the multiple ripples that the intersection’s and link’s lines exhibit in Figure 8. This effect is more prevalent on link agent teams (which are the most decentralized). In this way, they manage to avoid local maximums and therefore have the potential to achieve greater final performance. By contrast, centralized agents always take smaller steps in the direction of a local maximum. This difference in behavior is a contributing factor to the superior performance of decentralized teams.

Another observation is that the impact of increasing state resolution (i.e., “incorporating travel time”) differs from (MA-ES) to CCEA on the different agent definitions (see Figures 6, and 9). Specifically, we can notice that with CCEA (Figure 9), the intersection agent performs significantly better than the link agent on average, while when the time variable is added to the definitions, both teams are equally good. The same does not completely apply to the (MA-ES) algorithm, in which in simpler cases both intersection and link agents perform almost as well as each other. This means that the impact of the credit assignment problem is prevalent in all cases with both algorithms, since the link agent teams have a hard time coordinating all of their agents to a common goal. This, in conjunction with their inferior performance in terms of delays, shows that centralized solutions cannot efficiently explore the whole action space, especially in more complex environments.

Furthermore, we can see that on CCEA the more decentralized the MA teams the higher the performance, while on (MA-ES) the intersection agent teams (the middle ground in terms of centralization) are the best-performing ones, indicating that the impact of different agent definitions is definitely dependent on the learning algorithm used, verifying that the agent definition of the domain is an additional independent optimization variable.

In general, we can see that the difference between the worst and best performing agent definitions for every AGV number throughout the training epochs (Figures 7, 8) is much higher for CCEA in both average and max performances than that of (MA-ES). This indicates that not only do agent definitions impact learning algorithms’ performance in different ways, but also that their significance is also dependant on the learning algorithm used.

4.3 Testing Reinforcement Learning methods

Evolutionary Strategies is an evolutionary algorithm that simulates gradient ascent; inspired by this, we wanted to check how policy gradient algorithms and other celebrated MARL algorithms perform in this domain.

At first, we implemented the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [15] algorithm, which has proven to be quite efficient in a wide variety of domains [20]. We also implemented the decentralized I-DQN algorithm, where each agent has its own independent training neural network. However, as we can see in Figure 10 neither algorithm managed to train at all their independent or centralized agents, due to the unique characteristics of the warehouse traffic management domain. In general, most reinforcement learning algorithms, like MADDPG and I-DQN, need to have very specific rewards functions that will eventually train the agent policies. As we found after extensive experimentation with three different reward functions (incentivizing the AGVs to move on the shortest available path; estimating progress towards the AGVs destinations; and evaluating deliveries’ progress within specific time windows), there was not an obvious reward function resulting to the efficient training of the agents. Figure 10 depicts the best results obtained for those algorithms across all reward functions tested.

Specifically, we believe that the main problem with our domain (with regard to using a reinforcement learning method of any kind) is that the rewards can only be observed many steps after their execution, whereas these types of algorithms usually expect immediate results, i.e. the delayed rewards problem is severe [19]. Another interesting detail of many multiagent environments is that when the agent team tries to increase the total accumulated reward it is not easy to determine which agent was responsible for that improvement, this affects the complexity of how an action of each agent should be really rewarded—i.e., a "credibility assignment problem" [3] exists.

In conclusion, it is understandable why the RL algorithms in question failed to achieve desirable performance, notwithstanding the numerous different combinations of neural networks, exploration rates, and reward functions tested. This underscores the value of (MA-ES) and CCEA as alternative learning methods of choice, which can be applied in many complex domains that suffer from the problems mentioned above [2].

4Even though the tested RL algorithms did not work in this setting, interesting points regarding their potential pairing with factorized MAS settings were raised. For instance, I-DQN is only implementable with link (and with time) agent definitions, due to limitations to the size of the discretized action space. This means that if we had taken a different agent architecture as a constant, like intersection agents, I-DQN would have been unimplementable.
5 CONCLUSIONS AND FUTURE WORK

The design process of a MAS usually takes the definition of the agent as a constant and then selects an algorithm that it optimizes by tuning its hyperparameters and also applying domain-specific techniques suitable for that environment. That process is limiting for domains where the structure of the MA team is factorizable, where new possibilities are enabled. For example in our domain if the “link with time” agent definition was taken as a constant, then CCEA would have been in the same performance ballpark as MA-ES—and after some optimization, it might have delivered slightly better results. On the other hand, if the selected agent definition was the “intersection with time”, MA-ES would have performed significantly better.

In this work, we demonstrated that the impact of different agent definitions indeed differs based on the learning algorithm used for the training of the agents. Also, we showed that the use of different agent definitions has a greater impact on more complex environments, for example when the number of training parameters significantly increases. Moreover, our work confirms that many learning problems can be solved easier by using evolutionary algorithms, especially when there are delayed rewards from the environment. Last but not least, we demonstrated that multiagent Evolutionary Strategies is an excellent fit for warehouse traffic management domains, and it outperforms CCEA [7] in every possible scenario in this domain.

There is much space for future work on combining agent factorization with (multiagent) learning, both in warehouse traffic management and also in alternative real-world domains of interest. To begin, testing many more different agent definitions is in order, including hierarchical structures and factorizations with different communication channels on various domains and frameworks. A multitude of alternative learning algorithms, e.g., RL algorithms,
could also be used to identify the source of the impact of different agent definitions. Finally, different potential application settings include competitive domains like pricing strategies in markets \cite{agogino2004efficient} and potentially mixed domains.

REFERENCES

\begin{enumerate}
\item D. V. Dyke. 1994. Chapter 4: Applications of Distributed Artificial Intelligence in Industry. In Foundations of Distributed Artificial Intelligence. Industrial Technology Institute, Ann Arbor, Michigan, USA.
\item Ioannis Mallidis, Romnert Dekker, and Dimitrios Vlachos. 2012. The impact of greening on supply chain design and cost: a case for a developing region. Journal of Transport Geography 22 (2012), 118\textendash{}128.
\end{enumerate}