A comprehensive analysis of agent factorization and learning algorithms in multiagent systems

Kallinteris, Andreas; Orfanoudakis, Stavros; Chalkiadakis, Georgios

DOI
10.1007/s10458-024-09662-9

Publication date
2024

Document Version
Final published version

Published in
Autonomous Agents and Multi-Agent Systems

Autonomous Agents and Multi-Agent Systems (2024) 38:27
https://doi.org/10.1007/s10458-024-09662-9

A comprehensive analysis of agent factorization and learning algorithms in multiagent systems

Andreas Kallinteris[1] · **Stavros Orfanoudakis**[1,2] · **Georgios Chalkiadakis**[1]

Accepted: 16 June 2024

Abstract
In multiagent systems, agent factorization denotes the process of segmenting the state-action space of the environment into distinct components, each corresponding to an individual agent, and subsequently determining the interactions among these agents. Effective agent factorization significantly influences the system performance of real-world industrial applications. In this work, we try to assess the performance impact of agent factorization when using different learning algorithms in multiagent coordination settings; and thus discover the source of performance quality of the multiagent solution derived by combining different factorizations with different learning algorithms. To this end, we evaluate twelve different agent factorization instances—or *agent definitions*—in the warehouse traffic management domain, comparing the training performance of (primarily) three learning algorithms suitable for learning coordinated multiagent policies: the Evolutionary Strategies (*ES*), the Canonical Evolutionary Strategies (*CES*), and a genetic algorithm (*CCEA*) previously used in a similar setting. Our results demonstrate that the performance of different learning algorithms is affected in different ways by alternative agent definitions. Given this, we can conclude that many important multiagent coordination problems can eventually be solved more efficiently by a suitable agent factorization combined with an appropriate choice of a learning algorithm. Moreover, our work shows that ES and CES are effective learning algorithms for the warehouse traffic management domain, while, interestingly, celebrated policy gradient methods do not fare well in this complex real-world problem setting. As such, our work offers insights into the intrinsic properties of the learning algorithms that make them well-suited for this problem domain. More broadly, our work demonstrates the need to identify appropriate agent definitions-multiagent learning algorithm pairings in order to solve specific complex problems effectively, and provides insights into the general characteristics that such pairings must possess to address broad classes of multiagent learning and coordination problems.

Keywords Agent factorization · Multiagent coordination · Warehouse traffic management · Evolutionary strategies

Andreas Kallinteris, Stavros Orfanoudakis have contributed equally to this work.

Extended author information available on the last page of the article

Published online: 26 June 2024

1 Introduction

The majority of benchmarks and frameworks for multiagent systems (MAS) assume a predetermined system structure and agent concept, as seen in classic problems like the prisoner's dilemma and predator–prey ones. 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 [1]. Such domains include network routing [2, 3], autonomous warehouse traffic management [4, 5], and power plant control [6]. In the autonomous warehouse traffic management domain, a MAS architect can design a system with different agent control structures. One option is to assign a single agent to control each autonomous ground vehicle (AGV). Alternatively, the architect may choose to assign a single agent to control each warehouse corridor. Lastly, the entire warehouse may be placed under the control of a single agent, representing another possible design choice. In domains where it is possible to factorize agent definitions, multiple ways exist to decompose the problem into a multiagent system. These approaches may vary in terms of the observable state and permitted actions of each agent, as well as in terms of the interactions between agents.

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 [7]. Similarly, the factorization method proposed in [8] is free from structural constraints and takes a new approach to transform the original joint action-value function into an easily factorizable one. There has also been some research on the impact of changing the agent structure while keeping state resolution the same. For instance, [9] proposed a cooperative *Multiagent Reinforcement Learning (MARL)* method that utilizes a factored critic combining per-agent utilities into the joint action-value function by changing the agent structure. Similarly, *AgentMixer* [10] creates a joint fully observable policy as a non-linear combination of individual partially observable policies based on the concept of factorization. In general, agent factorization in MARL is proposed in order to effectively tackle the challenges of environmental instability and the exponential growth of action space in multiagent systems [11]. 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 [12, 13], which compare six different agent definitions using learning algorithms in a warehouse traffic congestion management domain.

Moreover, to the best of our knowledge, besides a conference paper of ours [13] that we extend here, there has been no research on the impact of agent definitions in conjunction with using different learning algorithms in the same domain. Studying the effects of modifying agent definitions using various learning algorithms is a task that is both challenging and important: It is challenging because decomposing the problem in a particular manner may pose significant challenges for certain learning algorithms, resulting in difficulty solving the problem. Therefore, there may not exist multiple agent definitions or factorizations of the multiagent team that are suitable for application across multiple learning algorithms. In complex domains, a particular multiagent (MA) learning algorithm typically requires a specific agent definition, which may correspond to a limited range of possible multiagent system (MAS) structures. It is possible that this set of structures may not overlap with the set required by another learning algorithm. Therefore, the appropriate MAS structure may vary based on the specific MA learning algorithm being used. At the same time, it is essential to combine and assess various agent definitions together with different learning algorithms. As such, assessing *pairings* of alternative agent definitions with different learning

algorithms is important, allowing for: (a) the identification of (the most) appropriate pairings to solve specific complex problems effectively; (b) the drawing of essential insights into the general characteristics that such pairings must possess to address broader classes of MA learning and coordination problems; and (c) the acquisition of further insights into the intrinsic properties of the learning algorithms that make them well-suited for specific domains.

As such, this work investigates the performance impact of combining agent factorization in combination with different learning algorithms in multiagent coordination scenarios. As a case study, we pick the trending multiagent warehouse traffic management problem [14]. This problem is particularly challenging for traditional reinforcement learning (RL) algorithms to solve due to its high dimensional state space, continuous action space, and sparse reward signals. We tested various learning algorithms in this domain, including the *cooperative co-evolutionary genetic algorithm* (*CCEA*) [15], which previous research has shown as being successful in this problem domain [4]; and the more elaborate Evolutionary Strategies (*ES*) optimization algorithm [16] that is known to constitute a scalable alternative to RL methods in high dimensional spaces with sparse rewards, while extending it to work for MA systems (*MA-ES*). Additionally, we extended the *Canonical Evolutionary Strategies* (*CES*) [17] algorithm (which is a variation of the classic ES [16] algorithm) in a straightforward manner so as it is applicable for MA systems, and evaluated this emerging *MA-CES* algorithm in this domain as well. Our findings and contributions are the following:

- The effect of different agent definitions on learning performance varies depending on the learning algorithm being used.
- The impact of using different agent definitions is more noticeable in *complex environments*, such as those with a significant increase in the number of training parameters.
- We provide intuitions on the pairings of agent factorization with learning algorithms, discussing how these are successful or incapable of tackling the problem at hand.
- Our experiments reveal that popular MARL algorithms such as Independent DQN [18] and MADDPG [19] are not capable of solving the warehouse traffic management problem.
- By contrast, evolutionary approaches perform significantly better in this complex domain, which requires handling delayed rewards and poses challenges related to credit assignment.
- Specifically, our experiments indicate that the MA-ES algorithm is an effective learning algorithm for the warehouse traffic management domain.

This paper constitutes an extension of our work in [13], appearing in the *Proceedings of the 12th Hellenic Conference on Artificial Intelligence* (SETN 2022). Several contributions are distinct in this paper, compared to those already in [13], namely:

- We introduced and tested 6 new agent definitions for the warehouse traffic management framework (doubling the number of tested agent definitions from 6 to 12).
- We evaluated our multiagent extension (*MA-CES*) of the *Canonical Evolutionary Strategies* algorithm, to demonstrate the impact of agent factorization with yet another algorithm (thus increasing the number of evaluated algorithms from 2 to 3).
- We evaluated and compared every learning algorithm and agent definition to obtain useful insights into the source of their performance impact.
- Additionally, we demonstrated the performance impact using each algorithmic variant's training convergence time.

Also, we present an extensive and thorough experimental evaluation of every combination of agent definition and learning algorithm in environments of varying complexity, and obtained significant intuitions that can be easily generalized. In summary:

- Arguably, both *Evolutionary Strategies* algorithms that were demonstrated in this study, and especially *MA-CES*, are capable of finding well-performing policies, while also achieving this in faster training times than *genetic algorithms*.
- Highly decentralized agent definitions have consistently lower convergence times when paired with *CCEA*, compared to the other agent definitions of *CCEA*. However, the decentralized agent definitions are rather slow when paired with the Evolutionary Strategies algorithms. This evidently supports our initial claim that the pairing of an agent definition with a learning algorithm has a significantly higher impact than just the agent definition itself.
- In environments representing more complex optimization problems, selecting an agent definition that encompasses a compromise between centralized and decentralized definitions—i.e., selecting a middle-ground solution—will give the best performance results.

The rest of this paper is structured as follows. Section 2 discusses the theoretical background and related work, presents our application domain, and provides a detailed explanation of the algorithms used. Section 3 describes our approach: Sect. 3.1 presents the already existing agent definitions along with the new agent definitions we introduce in this work; and then, Sect. 3.2 presents the multiagent learning algorithms we put forward. Section 4 provides our thorough experimental evaluation process. Finally, Sect. 5 concludes this paper and outlines directions for future work.

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 [1]. Properly factorized MAS architectures are necessary for domains where naturally occurring entities cannot be effectively modeled as individual agents. For instance, in a complex computer network routing domain, an appropriate factorization of the problem, potentially using a hierarchical approach, is necessary to obtain meaningful results. When factorizing a MA system, it is crucial to ensure that each agent controls a subset of the environment that corresponds logically to a set of connected entities. As a counterexample, consider the case of robot soccer, where a multiagent architecture with two agents, one controlling the left leg of each robot player and the other controlling the right leg of each robot player, would result in each agent controlling a set of entities that are not directly related in the environment.

Now, a fairly recent work on the impact of agent factorization is [12], 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 how the state-action information is split to create new agent teams. Indeed, to the best of our knowledge, paper [12] has so far been the

only paper[1] to have examined the impact of agent definitions/agent factorization on multiagent coordination in depth. Our work in this paper expands this idea to also include, along with agent factorization, the learning algorithm as a new important factor. We thus depart from the prior work's assumption that the learning algorithm is treated as a constant.

As such, in this article we study the performance impact of combining (mainly) 3 different learning algorithms with 12 different agent definitions; and as a result, come up with various interesting new observations that add to the lessons learned in [12] For instance, in [12], it was observed that centralized agents excel in less intricate scenarios, such as those involving 90-200 AGVs, while they underperform in more complex environments when employing a neuroevolution strategy. However, our research demonstrates that the utilization of an alternative evolutionary algorithm significantly alters the performance dynamics across various agent definitions. Moreover, as detailed in Sect. 3.1.2 below, our work augments that of [12] by introducing a new state representation that exploits additional state information to that used in [12].

Now, the optimization problem of warehouse traffic management has drawn significant attention in the literature regarding the operational strategies of autonomous ground vehicles (AGVs) [20, 21]. Regardless, the authors of [4] 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.

Genetic algorithms are known for their unique "Darwinian" approach and have been used to solve various optimization problems across multiple domains [22]. These algorithms are known for their efficient action exploration rate, which is achieved through the use of mutation and crossover steps, as well as a carefully chosen fitness function. The multiagent genetic algorithm used in [4] is known as *CCEA*, which was initially introduced by [15] to solve general optimization problems involving multiple functions.

2.1 Warehouse traffic management

Warehouses are essential in every logistic system in the world [23]. Nowadays, more and more warehouses require fewer human workers, especially in inter-warehouse logistics, because of the breakthrough of autonomous ground vehicles (AGVs) [24]. 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 [25].

In this study, we investigate the performance of different learning algorithms for coordinating autonomous ground vehicles (AGVs) in the warehouse traffic management domain. Specifically, we examine the effectiveness of twelve different agent definitions when combined with various learning algorithms. The domain was originally presented in [4]. There, the complexity of the domain was determined by the number of AGVs present, where a lower AGVs count corresponds to a simpler environment, and a higher count indicates a more complex setting.

The warehouse, shown in Fig. 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 $cap_e \geq n_e(t)$ representing its physical area in a warehouse, and AGV traverse

[1] The only exception is our [13] conference paper, which we extend here in the ways already outlined in the introduction.

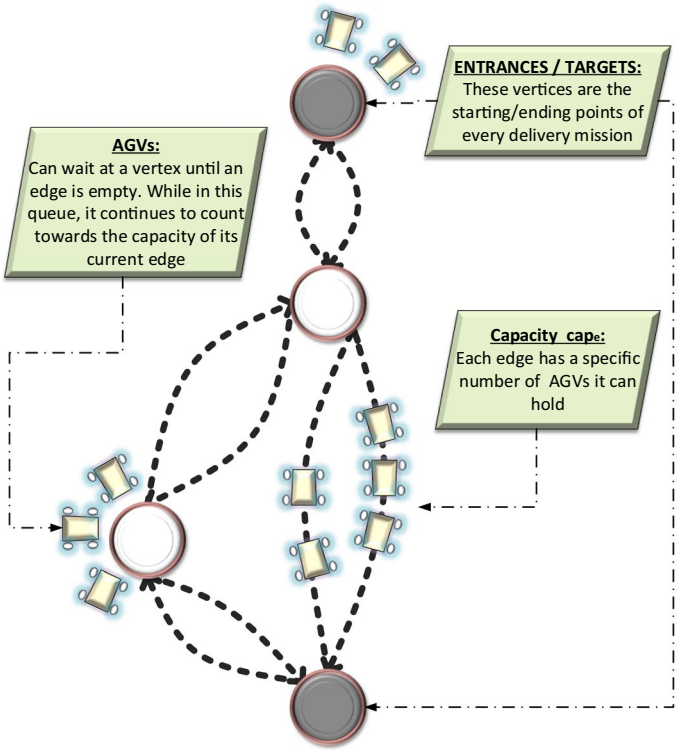

Fig. 1 Overview of the components of a simple warehouse and its core rules based on the work of [4, 13]

time $cost_e^{travel}$; with $n_e(t)$ being the number of AGVs traversing an edge e during timestep t, and the vertices $v \in V$ represent stock points that do not have a capacity–instead, when an AGV is waiting for 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

$$cost_e^{total}(t) = cost_e^{travel} + cost_e^{add}(t) \tag{1}$$

and it is observable by each AGV at (every) timestep t. The $cost_e^{add}$ in Eq. 1 is the additional cost computed by the optimization algorithm (see Sect. 2.2 below) used for minimizing traffic congestion. 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 traffic congestion. To this end, the traffic management system has to learn how to find the most suitable additional costs $cost_e^{add}(t)$ $\forall e \in E$ given the current state S of the warehouse, where the state consists of the current location of the AGVs and their delivery mission.

An optimal solution exists for every warehouse traffic congestion problem, but its exact computation is eventually rendered infeasible when the number of AGVs in a warehouse

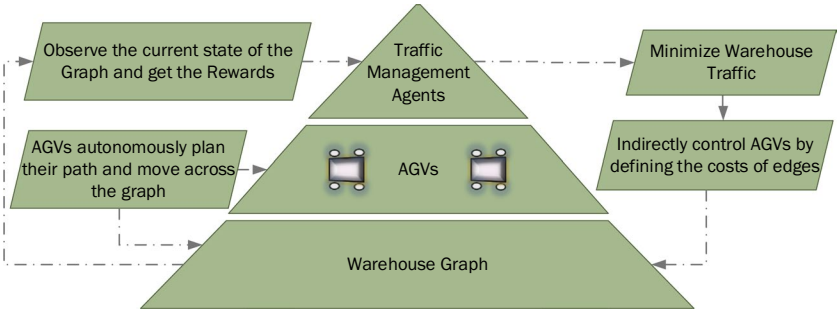

Fig. 2 Hierarchical planning as used in this warehouse domain [4, 13] in which actors (AGVs) do not communicate directly with the traffic management agents

increases. Traditional optimization techniques, such as those based on mathematical programming [26], often struggle to solve this problem fast enough because of its high dimensionality. This challenge persists even in relatively small warehouses with only a handful of AGVs. Furthermore, the need for real-time solutions adds to the difficulty, as traditional optimization methods typically lack the scalability required to address such demands. Thus, instead of using optimization techniques, a team of *traffic management agents* can be used, where each agent observes and controls a subset of the total state—and these subsets are non-overlapping. *This transforms the problem into one that calls for the learning of coordinated agent policies.*

The state space s_i for each agent i consists of (potentially incomplete) state-related information s_e from the set ε_i of edges that it controls, where $E = \bigcup_i \varepsilon_i$; and the action space a_i is the additional cost for those edges:

$$\begin{aligned} s_i(t) &= \left[s_e(t)\right], \quad \forall e \in \varepsilon_i, \\ a_i(t) &= \pi_i\left(s_i(t)\right) = cost_e^{add}(t), \quad \forall e \in \varepsilon_i \end{aligned} \tag{2}$$

Each agent i has a policy π_i that calculates the action that should be undertaken by the agent given a state s_i. Specifically, the role of the policy π_i is to directly output the additional (congestion-related) edge cost $cost_e^{add}$ $\forall e \in \varepsilon_i$, where $cost_e^{add} \in \mathcal{R}$ can be any real number[2] (i.e., the action space is continuous). Naturally, the goal of the agent team (i.e., the multiagent system) is to maximize the number of total deliveries G by balancing the $cost_e^{add}$ computed by the agents $\forall e \in E$, in a way that will minimize congestion. Figure 2 summarizes how the multiagent system (i.e., the set of warehouse traffic management agents) and the AGVs interact with each other to find the optimal solution.

2.2 Learning algorithms

The learning process of a multiagent team in a warehouse traffic management domain is complex. In many cases, classic reinforcement learning algorithms are usually challenged by the complexity of this problem. To train autonomous agents to control a warehouse

[2] Notice that $cost_e^{add}$ can take values lower than 0; in such cases, this agent-computed $cost_e^{add}$ essentially incentivizes the AGVs to traverse the particular edge e.

efficiently while avoiding harmful collisions, Evolutionary Algorithms [27] or reinforcement learning methods can be explored, as they show considerable promise for addressing these types of challenges. In this section, we present several algorithms we adopted or extended for use in this study, starting from genetic algorithms [28], continuing with Evolutionary Strategies [27], and concluding with certain well-known multiagent RL Algorithms [29, 30].

2.2.1 Genetic algorithms

Genetic algorithms [28] are widely used optimization algorithms known for their effectiveness in solving complex real-life tasks such as optimizing the assignment of shifts in a simple setting or the optimization of autonomous warehouse traffic in a more complex scenario. The fundamental concept behind genetic algorithms involves the generation of a random population of genes which undergo successive mutations and crossovers using the genes with the highest fitness scores until the gene with the best fitness score is obtained.

Algorithm 1 *Genetic algorithm*

1 $pop_{(0)}$ = new randomly initialized population
2 **for** *each iteration* $t = 0, 1, 2, \ldots$ **do**
3 Evaluate *Fitness* of each gene in $pop_{(t)}$
4 Select k genes with the highest *Fitness* value
5 $pop_{(t+1)}$ = Mutate and Crossover the k genes

Algorithm 1 illustrates the fundamental steps of the genetic algorithm, which is a popular optimization method for solving complex real-life tasks [28]. The algorithm starts by randomly generating a population of *N* genes in line 1, where each gene represents a single parameter or a set of parameters that need to be optimized. Next, in line 3, the fitness function evaluates each gene based on domain-specific properties, determining its fitness score. The fittest genes are then selected for mutation and crossover in lines 4–5. Typically, only the top *k* genes are selected for these operations, while the rest are discarded. The mutation operation randomly changes the value of a gene, while the crossover operation combines two or more genes to create a new one. This process of selection, mutation, and crossover is repeated until a satisfactory level of optimization is achieved.

2.2.2 Cooperative coevolutionary algorithm

The multiagent genetic algorithm we experimented with is the *Cooperative Coevolutionary Algorithm*. The *CCEA* [15] algorithm is a multiagent genetic algorithm that aims to coordinate the actions of multiple agents to solve a joint problem. Unlike a standard genetic algorithm, the fitness function used in *CCEA* reflects the total utility of all coordinating agents at the end of a simulation, rather than individual fitness scores for each agent. This means that the fitness function used in *CCEA* must be a joint fitness function that models the coordination problem.

Algorithm 2 *CCEA*

1 **for** *each agent a* **do**
2 $pop^{a}_{(0)}$ = new random population
3 **for** *each iteration* $t = 0, 1, 2, \ldots$ **do**
4 **for** *each agent a* **do**
5 Evaluate *Fitness* of the genes in $pop^{a}_{(t)}$
6 Select k genes with the highest *Fitness* value
7 $pop^{a}_{(t+1)}$ = Mutate the k genes by adding Gaussian noise

As seen in Algorithm 2, the approach's core is that of a *genetic algorithm*. Therefore, similarly to 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 mutated so that a new population of *N* genes will be created. In the context of this study, *CCEA* uses parameterized neural networks (NN) to represent each gene. This way, the mutation is performed by adding noise, sampled from a Gaussian distribution, directly to the neural network's weights.

2.2.3 Evolutionary strategies

Evolutionary Strategies [27] share many similarities with traditional genetic algorithms and are part of the broader family of evolutionary computation techniques. These optimization methods rely on a large set of offspring, each with diverse traits, although their distinguishing features are primarily determined by the approach employed to generate subsequent generations. This paper will focus on a modern variant of Evolutionary Strategies [16]. 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 α, noise standard deviation σ, population size n
1 θ_0 = randomly initialized policy
2 **for** *each iteration* $t = 0, 1, 2, \ldots$ **do**
3 Sample $\phi_1 \ldots \phi_n \sim \mathcal{N}(0, 1)$
4 $F_j = F(\theta_t + \sigma\phi_j)\ \forall$ samples $j \in [1, n]$
5 Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{j=1}^{n} F_j \phi_j$

Algorithm 5 outlines the steps of the Evolutionary Strategies algorithm. Firstly, in line 1, a random base policy θ_0 is initialized. Next, in line 3, for each iteration *t*, the algorithm generates *n* samples from a Gaussian distribution and adds them as noise to each policy gene. Like CCEA, Evolutionary Strategies use parameterized NNs; therefore, the Gaussian noise is directly added to the NN's weights θ. These perturbed policies are then evaluated

using the fitness function. Finally, the new policy θ_{t+1} is generated by taking a weighted sum of the previous policy θ_t and the sampled policies from the current iteration, where the weights are based on the fitness score of each sample, as depicted in line 5. This process gradually leads the policy θ to converge to a solution with the best fitness score.

2.2.4 Canonical ES

Another variation of the classic Evolutionary Algorithms [31] is the *Canonical Evolutionary Strategies* algorithm (CES) [17, 31]. The main difference between this algorithm and the simple *ES* that is presented above is in the final part, where the weights for the gradient update are calculated. The ES algorithm, as depicted in Algorithm 5, utilizes the policy networks θ_i of every different sample weighted on their final fitness F_i. On the other hand, while CES maintains the same principle of aggregating the information from every sample ϕ_i, it creates an alternative weighting function based on a sorted list of the samples' fitness scores.

Algorithm 4 *Single-agent canonical evolutionary strategies*

Data: Learning rate α, noise standard deviation σ, population size n
1 $\theta_0 =$ Randomly initialized policy
2 $w_j = \frac{log(n+0.5)-log(j)}{\sum_{k=1}^{n} log(n+0.5)-log(k)}, \forall j \in [1, n]$
3 **for** *each iteration* $t = 0, 1, 2, \ldots$ **do**
4 Sample $\phi_1 \ldots \phi_n \sim \mathcal{N}(0, 1)$
5 $F_j = F(\theta_t + \sigma\phi_j)\ \forall$ samples $j \in [1, n]$
6 Sort ϕ_j's according to F score to obtain a $(\hat{\phi}_1, \ldots, \hat{\phi}_n)$ sorted vector ($\hat{\phi}_1$ is ϕ_j with best F score)
7 Set $\theta_{t+1} \leftarrow \theta_t + \sigma \sum_{j=1}^{n} w_j \hat{\phi}_j$

Algorithm 4 shows this *Canonical ES* approach. Similarly to *ES* depicted in Algorithm 5, the first step is to instantiate the random policy parameters θ_0. Then every iteration includes four steps. Initially, fetch n samples from a standard Gaussian distribution $\mathcal{N}(0, 1)$, and modify them to generate n new genes. The values of the perturbed genes are added to the policy parameters θ_t to generate n new policies, where each one is evaluated based on a fitness function $F()$. The third step of this algorithm includes sorting the n new samples in descending order based on their fitness score F_j (samples that lead to higher fitness scores are first). Finally, the approximated gradient update $\sigma \sum_{j=1}^{n} w_j \phi_j$ is calculated by multiplying[3] the weights vector w_j with the sorted samples $\hat{\phi}_j$.

Therefore, the main characteristic of the Canonical *ES* algorithm revolves around the weight vector w_j. In detail, the normalized weight vector w_j has n values, one for each gene in the population. This vector takes values in [0,1] where the weights with a smaller index–thus higher fitness score—have greater values than the rest. Also, since this weight vector is a normalized weighting function, it is designed to sum up to 1, to ensure the

[3] This is an element-wise multiplication.

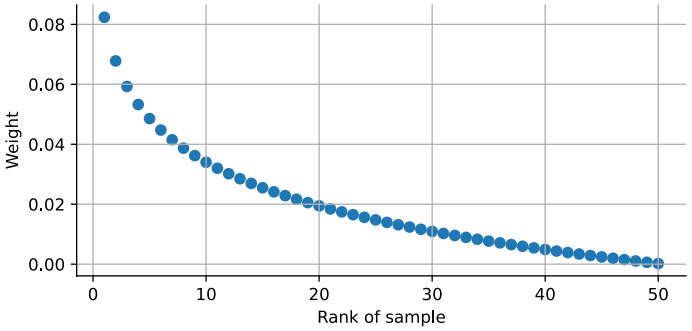

Fig. 3 The discrete normalized weight function w_j of the Canonical ES algorithm for population size $n = 50$. Notice that for samples closer to 0, the weight value increases logarithmically

generated gradients will be bounded, thus reducing the possibility of exploding gradients [32]. Figure 3 demonstrates the values of the Canonical ES weight vector for a population with 50 genes. Notice that the gene with the highest fitness score, i.e., w_1, has a much larger value than the rest. This leads to gradient updates that are based more on which sample performed better than on how much better it actually performed. By contrast, the ES algorithm calculates its gradient update based on how much better each sample actually performed.

2.2.5 Independent deep Q networks (I-DQN)

In (single or multiagent) reinforcement learning, an agent receives a reward that can be positive or negative for every action taken. The agents use this reward signal to learn how to approximate the expected cumulative reward r by choosing the action with the highest Q-value. Traditional Q-learning algorithms maintain a lookup table $Q(s, a)$ to store the value function estimate for each state-action pair. However, as the state and action spaces grow, the size of the table also grows exponentially, making it infeasible to maintain.

Deep Q-Networks (DQN) [18], a popular deep reinforcement learning algorithm, combines Q-learning with neural networks to approximate the value function. Here, the Q-value is represented by a deep neural network, which takes in the state as input and outputs the Q-value for each possible action. The neural network is trained using stochastic gradient descent to minimize the mean-squared error between the predicted Q-values and the target Q-values, which are computed using the Bellman equation [30]. This allows for function approximation and enables DQN to handle large state and action spaces. Extending DQN to multiple agents, calls for using a neural network for each agent, in order to evaluate the score of the different state and action combinations.

2.2.6 Multi-agent deep deterministic policy gradient

MADDPG [19] is a multiagent policy gradient algorithm that builds on the (single-agent) *Deep Deterministic Policy Gradient (DDPG)* [33] RL algorithm.[4] In MADDPG, each agent learns a centralized critic by considering the observations and actions of all agents. The critic is trained to exploit the policies of other agents, while the actor is trained to use only local information. This approach enables the learned policies to use only the agent's own observations during execution, making it applicable to both cooperative and competitive environments. During execution, only the local actors are used, operating in a decentralized manner, while the critics are solely utilized for training purposes.

2.3 Related work

In this section, we present an overview of interesting work relating to state-action space factorization and autonomous warehouse optimization.

2.3.1 State-action space factorization

Planning and coordination problems often have a structure in their reward and value functions, state transitions, and observations. Also, there is a structure in the relationships among features used to describe states, actions, rewards, and observations. This structure can be exploited by specialized representations and algorithms for computational leverage, resulting in increased system performance and flexibility, and allowing for the efficient representation of complex information [12, 37].

There are many algorithms and tools developed to utilize the state-action space factorization property of various important problems. For instance, [38] proposes a strategy to address challenges in Deep Reinforcement Learning (DRL) by demonstrating the importance of choosing appropriate state and action spaces for the problem at hand. The authors build a DRL agent using Deep Q Networks (*DQN*) [18] on a building simulator, and compare it with a widely used baseline heuristic method.

Many MARL approaches belonging in the centralized training with decentralized execution (CTDE) regime, such as *VDN* [39] and *QMIX* [40], factorize the joint action-value function into individual ones for decentralized execution. However, they are often limited by structural constraints such as additivity and monotonicity. The authors of [8] propose a new factorization method called *QTRAN* that is free from such constraints and transforms the original joint action-value function into an easily factorizable one. Furthermore, in [41], the effects of centralized and decentralized critic approaches are formalized and experimentally evaluated in multiple environments to provide a deeper understanding of the implications of critic choice.

Additionally, *FACMAC* [9] is a cooperative multiagent reinforcement learning approach that learns policies using deep deterministic policy gradients. It utilizes a centralized but factored critic, which combines per-agent utilities into the joint action-value function via a non-linear monotonic function, similar to the *QMIX* algorithm.

[4] Concluding this section, it is worth mentioning that there is a line of work that combines neuro-evolution with (deep) reinforcement learning for optimizing the performance of a policy [34]. For instance, [35] explored combining a simple cross-entropy method with DDPG [33] or TD3 [36].

However, unlike *QMIX*, *FACMAC* employs a non-monotonic factorization that allows for increased representational capacity, which enables it to solve some tasks that cannot be solved with monolithic or monotonically factored critics. Furthermore, [42] introduces a multiagent learning approach that integrates relationship awareness into value-based factorization methods.

2.3.2 Autonomous warehouse optimization

The optimization of autonomous warehouses is a complex task that requires careful planning. In this paper, we have presented methods that try to find the optimal solution to this problem by employing genetic and evolutionary algorithms; however, other methods can also help solve this particular task. Lee et al. [43] focused on optimizing mobile robot paths in a warehouse environment with automated logistics using reinforcement learning. To achieve this, the researchers compared the results of experiments conducted using two basic algorithms, which helped identify the fundamentals required for planning the path of a mobile robot and utilizing reinforcement learning techniques for path optimization. These algorithms were tested using a path optimization simulation of a mobile robot in the same experimental environment and conditions.

Additionally, [44] presents a fully automatic solution for optimizing product placement and order-picking routes in a warehouse using genetic algorithms. The solution takes into account the warehouse structure and list of orders to minimize the sum of order picking times. Techniques such as multi-parent crossover, permutations, and local search are used to optimize order-picking routes. Then, a genetic algorithm is used to optimize product placement with ideas such as caching, multiple restart, and order grouping to improve and accelerate optimization.

Another paper [45] explores the use of automated vehicle systems for optimizing warehouse operations and reducing costs. The focus is on strategies for minimizing travel time and avoiding congestion, as well as solving storage assignment problems. The paper proposes storage assignment strategies and route planning of AGVs in a traditional rectangular warehouse layout. Moreover, [46] proposes a distributed framework for determining the shortest path for a swarm of robots in a warehouse for logistic applications. The proposed solution uses optimization routines to avoid downtime and collisions between robots and is based on the reference model using Dijkstra, Floyd-Warshall, and Bellman-Ford algorithms.

Furthermore, some works study improvements and techniques that can be applied to autonomous warehouses. [47] discusses the challenges associated with processing and analyzing the increasing amount of unstructured data, and the need to upgrade existing Data Warehouses to use this data effectively. The proposed solutions are indexing and sorting, which are intermediate solutions for modern data processing technologies. By adopting these methods, organizations can improve the return on investment from their existing data warehouse environment. Raghuram et al. [48] examine how warehouse pickup and delivery schedules are affected by various factors and how data analytics can be used to process demand data to optimize warehouse layout and pick locations. The authors analyzed the layout of an electronic warehouse handling over 10 million daily orders and made modifications that resulted in a significant reduction in travel distance and workforce. They suggest that periodic demand data analysis can lead to reduced lead times and costs.

3 Our approach

In this chapter, we first describe how we factorized the warehouse traffic management domain to obtain appropriate agent definitions; and then proceed to present the multiagent learning algorithms we modified to evaluate the performance impact their pairing with the agent definitions we designed has on multiagent coordination problems.

3.1 Agent definitions

In the domain of autonomous warehouse traffic management, it is possible to partition the traversal graph into subsets that can be controlled by individual agents. This presents an opportunity to examine the influence of distinct agent definitions in combination with a range of parameters, such as the learning algorithm employed, as well as the level of complexity associated with the optimization task, relative to the number of automated ground vehicles (AGVs) utilized.

In this section, we define 12 different agent definitions, where 6 of them were first introduced in [12], which vary in:

1. *Agent structure*: how 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.

3.1.1 Modifying agent structure

Initially, we will present three agent factorizations based on different partitions of the total agent structure.

Centralized agent

The first variant we are investigating is conceptually the simplest, as it involves assigning a single agent to manage all the edges in the graph ($\|I\| = 1$). The *centralized* agent's state s is defined as the total number $n_e(t)$ of AGVs currently traversing each edge, while the *centralized* agent's possible actions are to assign the additional cost of travel $cost^{add}$ for every edge (Eq. 3).

$$\begin{aligned} s^{Central}(t) &= \left[n_e(t)\right] \quad \forall e \in E, \\ a^{Central}(t) &= \left[cost_e^{add}(t)\right] \quad \forall e \in E, \end{aligned} \tag{3}$$

Figure 4 depicts how a potential warehouse graph is partitioned to allow a single *centralized* agent to control all the edges. In detail, a *centralized* agent controls all the edges with the same color, i.e. the whole traversing graph. From the agent's perspective, this is a very complex, high-dimensional learning problem; however, from the "agent team's" perspective, it is the simplest since there is no other agent to cooperate with.

Link agent

We proceed to examine the second variant of the agent structure factorizations, which stands in contrast to the *centralized* agent. In this case, each *link* agent i is responsible for

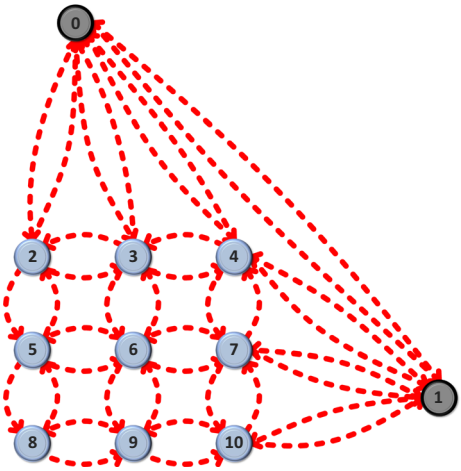

Fig. 4 A visualization of the *centralized* agent for a simple warehouse graph, where the edges represent the corridors and the nodes the intersections of a warehouse. Here, the agent manages the entire warehouse

controlling a single directed edge e_i. Consequently, the multiagent team is comprised 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 $cost^{add}_{e_i}$ (Eq. 4).

$$\begin{aligned} s_i^{Link}(t) &= n_{e_i}(t), \\ a_i^{Link}(t) &= cost^{add}_{e_i}(t), \end{aligned} \tag{4}$$

Figure 5 visually presents the edges such a *link* agent controls. This agent definition creates the most decentralized teams, resulting in the most challenging multiagent team coordination setup. Since it is the most decentralized agent definition, it is the one affected the most by the structural credit assignment problem [25]. In particular, the agents receive the total team performance as a learning signal, without indicating individual contributions. In more complex autonomous warehouse optimization tasks, where there are more agents, contribution becomes more ambiguous and noisy. However, each individual agent is very simple, since $\|s_i^{Link}\| = \|a_i^{Link}\| = 1, \forall i \in I$ making its training straightforward.

Intersection agent

The final variant based on the agent structure we investigate is a middle ground between the completely centralized and completely decentralized *link* team. Here, each of these agents is responsible for controlling the AGV traffic on the set of all incoming edges ε_i of a vertex v of the graph G. So, the team consists of $\|I\| = \|V\|$ *intersection* agents, whose states and actions are presented in Eq. 5.

$$\begin{aligned} s_i^{Intersection}(t) &= [n_e(t)] \quad \forall e \in \varepsilon_i, \\ a_i^{Intersection}(t) &= [cost_e^{add}(t)] \quad \forall e \in \varepsilon_i, \end{aligned} \tag{5}$$

Figure 6 provides a visual representation of a warehouse graph controlled by an *intersection* agent team. Notice that in Fig. 6, each color represents the edges controlled by a single agent. The state-action space for each *intersection* agent i is, $\|s_i^{Intersection}\| = \|a_i^{Intersection}\| = \|\varepsilon_i\|$ which is very likely to create heterogeneous agents, since not all vertexes have the same number of incoming edges. Furthermore, the number of *intersection*

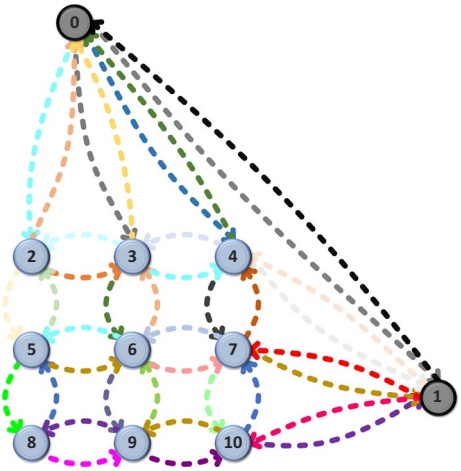

Fig. 5 A visualization of the *link* agent team, each *link* agent manages a single edge (link) of the warehouse

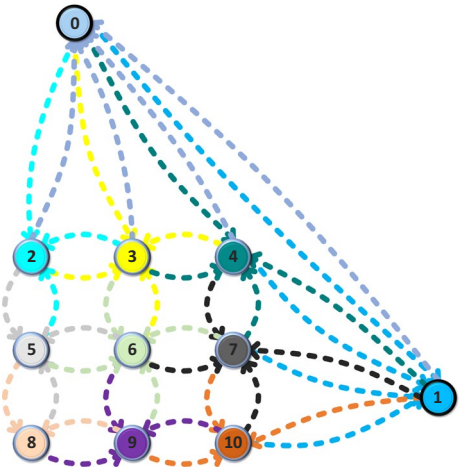

Fig. 6 A visualization of the *intersection* agent team, each *intersection* agent manages the set of incoming edges (intersection) of the warehouse

agents is less than that of *Link* agents 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.

3.1.2 Modifying state resolution

All the previous variants focused on the effect of agent team structure on the multiagent learning problem without altering the state resolution. Up until now, the state information only consisted of the current number $n_e(t)$ of AGVs traversing the edges. Here, we will propose three variations of the existing agent structure factorizations based on different definitions of the state resolution of every individual agent. The following variants incorporate AGV travel time information to investigate further the effect of including additional AGV

tracking data, thus providing more data to the learning algorithm and making the optimization task more complex.

Incorporating last AGV travel time

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, i.e., in how many steps can an AGV leave the edge i (and thus $d_e(t)$ is an indicator of when the n_e component of the state is about to change). In detail, $d_e(t)$'s value can range from the total time required to traverse an edge down to 1, or when the edge is empty to 0. As $d_e(t)$ corresponds to the time required for an AGV to become the last AGV from e to reach the stock point (node) u reachable from this edge, we perhaps (not very appropriately) term the $d_e(t)$ as the "last AGV travel time"[5] for this edge. Equation 6 depicts the modified state resolutions for each agent structure we defined previously.

$$\begin{aligned} s^{Central,LastTime}(t) &= \left[n_e(t), d_e(t)\right] \quad \forall e \in \varepsilon \\ s_i^{Link,LastTime}(t) &= \left[n_{e_i}(t), d_{e_i}(t)\right], \\ s_i^{Intersection,LastTime}(t) &= \left[n_e(t), d_e(t)\right] \quad \forall e \in \varepsilon_i, \end{aligned} \tag{6}$$

Note that the action space for each agent definition remains the same as in Eqs. 3, 4, and 5, respectively. These agent definitions enable us to investigate the impact of increasing the observation complexity on the previous agent structures.

Incorporating average AGV travel time

Furthermore, we experimented with additional state information. Here, the observable state of each edge is augmented by $j_e(t)$, which represents the average amount of time remaining until all the AGVs complete their traversal on that edge, $j_e(t)$'s value can range from the total time required to traverse an edge down to 1, or when the edge is empty to 0. Equation 7 shows how the state space will transform for each corresponding agent structure.

$$\begin{aligned} s^{Central,AvgTime}(t) &= \left[n_e(t), j_e(t)\right] \quad \forall e \in \varepsilon \\ s_i^{Link,AvgTime}(t) &= \left[n_{e_i}(t), j_{e_i}(t)\right], \\ s_i^{Intersection,AvgTime}(t) &= \left[n_e(t), j_e(t)\right] \quad \forall e \in \varepsilon_i, \end{aligned} \tag{7}$$

These agent definitions have the same state complexity as *Last* AGV Travel Time, but here they observe a different part of the environment.

Incorporating both average and last AGV travel time

Lastly, we investigate variants that augment the observable state of each edge with both the last travel time $d_e(t)$ and average remaining travel time $j_e(t)$, as shown in Eq. 8.

$$\begin{aligned} s^{Central,BothTimes}(t) &= \left[n_e(t), d_e(t), j_e(t)\right] \quad \forall e \in \varepsilon \\ s_i^{Link,BothTimes}(t) &= \left[n_{e_i}(t), d_{e_i}(t), j_{e_i}(t)\right], \\ s_i^{Intersection,BothTimes}(t) &= \left[n_e(t), d_e(t), j_e(t)\right] \quad \forall e \in \varepsilon_i, \end{aligned} \tag{8}$$

[5] $d_e(t)$ had already been incorporated in the state information in [12], where it was referred to simply with the term "travel time". We refer to it by a different term in our paper, to clearly differentiate from the *average AGV travel time* which we introduce in our work.

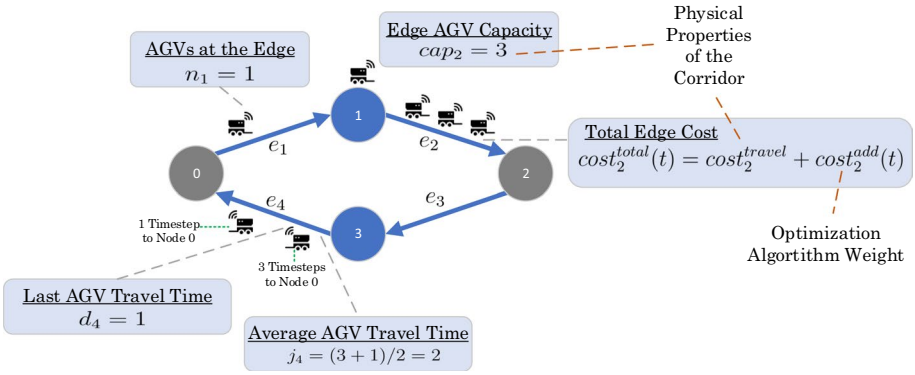

Fig. 7 A visual representation illustrating the key variables within a warehouse featuring four corridors (edges). These variables include the edge AGV capacity (cap_e), travel cost ($cost_e^{travel}$), and AGV travel times (d_e, j_e), all contingent upon both warehouse-specific factors like corridor size and AGV attributes such as ground speed

The agent definitions incorporating both *Average* and *Last* AGV Travel Time have the largest observation complexity and are expected to provide us useful insights on the performance impact of combining learning algorithms with more complex agent definitions.

3.1.3 Warehouse example

After detailing all the warehouse variables, it is beneficial to provide an example involving a specific instantiation of our domain. Figure 7 depicts an example of a 4-corridor warehouse and its parameters as described in Sect. 2.1. Specifically, we can see that the second edge (e_2) is currently traversed by three AGVs ($n_2 = 3$), while one AGV is waiting in node 1 because edge 2 cannot physically hold more AGVs ($cap_2 = 3$ AGVs). Moreover, there are two AGVs in edge 4 where the first needs 1 timestep to reach node 0 while the other needs 3 timesteps to do so. This information helps create the additional state space information (see Sect. 3.1.2). In detail, the "last AGV travel time" d_e mentioned in Sect. 3.1.2 is the steps required for an AGV to leave the edge, thus $d_4 = 1$ in our case, because the AGV needs 1 timestep to exit the edge. The "average travel time" is the average time needed for the AGVs in an edge to fully traverse it; hence, $j_4 = 2$ timesteps. The total cost of traversing an edge $cost_e^{total}$, as defined in Eq. 1, is the sum of the actual cost of traversing the edge $cost_e^{travel}$ (representing the length of the corridor) and the added cost $cost_e^{add}$ which is the output of the optimization algorithm used.

3.1.4 Agent definitions summary

We have designed various agent factorizations by dividing the autonomous warehouse domain into many pieces. In practice, we have combined the agent factorizations based on agent structure and those based on alternative state resolution to create 12 new agent factorizations. In a previous study [12], six agent definitions were used, including centralized, intersection, and link, along with their corresponding variants that incorporate average travel time. Table 1 demonstrates the 12 agent factorizations we have formulated to review the performance impact of combining specific agent factorization with different learning algorithms in a challenging multiagent coordination setting.

Table 1 This table displays the 12 agent factorization (left column) we designed

Agent definition	Agent structure			State resolution	
	Centralized agent	Link agent	Intersection agent	Last AGV travel time	Average AGV travel time
Centr	★				
Centr. last time	★			★	
Centr. avg time	★				★
Centr. both times	★			★	★
Inter			★		
Inter. last time			★	★	
Inter. avg time			★		★
Inter. both times			★	★	★
Link		★			
Link last time		★		★	
Link avg time		★			★
Link both times		★		★	★

Every agent definition created has a combination of a single agent structure (centralized, link, intersection) and none, one, or both of the additional state information available (*last* AGV and *average* AGV travel time). The ★ indicates that the agent definition has that property

3.2 Learning coordinated multiagent policies

This section provides an overview of the multiagent algorithms that were created and utilized to investigate the impact of different learning algorithms on training performance in the warehouse traffic management domain, while also exploring various agent definitions.

3.2.1 Multiagent evolutionary strategies

As previously mentioned, various methods exist to generate subsequent generations of genes. However, stochastic gradient ascent in particular is renowned for its adaptability to noise and its robustness in exploring large search spaces; it is particularly effective when dealing with complex, high-dimensional problems where deterministic methods may struggle. As such, the present section will concentrate solely on a technique utilizing simulated stochastic gradient ascent, as outlined in [16].

Specifically, in a learning problem, $F()$ will be the stochastic fitness function influenced directly by the environment, and θ 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 θ, and which can be approximated with sampling as:

$$\nabla_\theta E_{\phi\sim N(0,1)} F(\theta+\sigma\phi) = \frac{1}{\sigma} E_{\phi\sim N(0,1)}\{F(\theta+\sigma\phi)\phi\} \tag{9}$$

In Eq. 9, ϕ is a random sample of a Gaussian noise distribution, and σ is the standard deviation. Using the gradient in Eq. 9 as the objective described in terms of policy θ,

Algorithm 5, *MA-ES*,[6] extends the *ES* [16] algorithm to multiagent settings in a straightforward manner. Specifically, MA-ES is an extension of ES in which, instead of evolving the policy of only an agent per epoch, it evolves the policies of many agents.

Algorithm 5 *Multiagent evolutionary strategies (MA-ES)*

Data: Learning rate α, noise standard deviation σ, $I = \{i_0, i_1, ...\}$ the set of agents, population size n

1 **for** *each agent* i **do**
2 $\quad \theta_0^i =$ randomly initialized policy
3 **for** $t = 0, 1, 2, \ldots$ **do**
4 $\quad$ Sample $\phi_1 \ldots \phi_n \sim \mathcal{N}(0, 1)$
5 $\quad F_j = F(\{\theta_t^0.\theta_t^1, .., \theta_t^{\|I\|-1}\} + \sigma\phi_j)\ \forall$ samples $j \in [1, n]$
6 $\quad$ **for** *each agent* i **do**
7 $\quad\quad$ Set $\theta_{t+1}^i \leftarrow \theta_t^i + \alpha \frac{1}{n\sigma} \sum_{j=1}^{n} F_j \phi_j$

MA-ES consists of many iterations with two phases each. Initially, every agent *i* gets a randomly initialized policy θ_0^i (line 2). Then, as displayed 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).

3.2.2 MA canonical evolutionary strategies (MA-CES)

ES algorithm is a black-box optimization algorithm suitable for finding solutions to complex optimization problems [27]. As we have seen, it calculates the policy gradient update by obtaining a weighted sum of the samples based on the final fitness score. By contrast, the *Canonical ES* algorithm (See Sect. 2.2.4) calculates the policy gradient update by multiplying a predefined weight vector w_i with the sorted list of samples based on their fitness score [17]. Therefore, the main difference between these two algorithms (*ES* and *CES*) is in the way they combine the knowledge gained by their genes. We could assume that there are specific problems in which each algorithm can be superior to the other. As we will see later, the multiagent version of the Canonical ES algorithm performs better in most cases for the autonomous warehouse congestion problem; however, MA-ES may provide better results for different cases.

Here, we put forward a straightforward extension of the classic single-agent *Canonical ES* algorithm for multiagent systems similar to *MA-ES*, as depicted in Algorithm 6. In particular, in a multiagent problem with *I* agents, each agent has a randomly initialized policy, θ_0^i potentially representing the parameters of a neural network. After that, an iterative process starts until a satisfying solution is found or a time limit is reached. An iteration starts by getting *n* samples, one for each gene in the population, from a Gaussian distribution $\mathcal{N}(0, 1)$. Then

[6] *MA-ES* with a single agent is the same as *ES*.

the new perturbed policies θ_t^i for each agent and every gene are calculated by simply adding the sampled trajectories ϕ_j to the policy networks' current values (line 6). By doing that, the algorithm aims to sufficiently explore as many neighbor states as possible, by simulating the outcomes given the new modified policies; hence bigger population sizes may lead to a better exploration rate.

Algorithm 6 MA canonical evolutionary strategies (MA-CES)

Data: Learning rate α, noise standard deviation σ, population size n, $I = \{i_0, i_1, ...\}$ the set of agents

1 **for** *each agent* i **do**

2 $\quad \theta_0^i =$ randomly initialized policy

3 $w_j = \frac{log(n+0.5)-log(j)}{\sum_{k=1}^{n} log(n+0.5)-log(k)}, \forall j \in [1, n]$

4 **for** *each iteration* $t = 0, 1, 2, \ldots$ **do**

5 $\quad$ Sample $\phi_1 \ldots \phi_n \sim \mathcal{N}(0, 1)$

6 $\quad F_j = F(\{\theta_t^0.\theta_t^1, .., \theta_t^{\|I\|-1}\} + \sigma\phi_j)\ \forall$ samples $j \in [1, n]$

7 $\quad$ **for** *each agent* i **do**

8 $\quad\quad$ Sort ϕ_j's according to F score to obtain a $(\hat{\phi}_1, \ldots, \hat{\phi}_n)$ sorted vector ($\hat{\phi}_1$ is ϕ_j with best F score)

9 $\quad\quad$ Set $\theta_{t+1}^i \leftarrow \theta_t^i + \sigma \sum_{j=1}^{n} w_j \hat{\phi}_j$

An important step of every iteration is the simulation. There, every set of agent policies $\{\theta_t^0.\theta_t^1, .., \theta_t^{\|I\|-1}\}$ is put to the test in the actual problem environment so that it can be evaluated using a fitness function *F*(). The fitness score F_j of every agent is then used to sort the sampled trajectories ϕ_j. For each agent, the trajectories ϕ_j that lead to higher fitness scores are placed first in a descending fashion to be multiplied by the predefined weight vector w_j. The final step of the *MA-CES* algorithm calculates the gradient update for every agent independently. Observe in line 9 that it is not required to normalize the gradient by dividing it by the population size, as it was previously done in *MA-ES* since it is already normalized because of the weight vector.

Notice that, even though each policy set $\{\theta_t^I\}$ shares a common sampled trajectory ϕ_j, the final policy gradient update is calculated separately for each agent in the team. This way, the algorithm ensures that every agent will benefit from both the local understanding of the agent and the global observations accumulated by the agent team across the whole population *n*.

4 Experimental evaluation

In this section, we provide a thorough experimental evaluation of our approaches along with details about our experimental setup, in several environments of varying complexity.[7] We used the warehouse design displayed in Fig. 8. The same warehouse was previously used in [4, 13] to study the impact of different agent definitions on learning performance.

We run many experimental runs for every different experimental scenario (agent definition, number of AGVs, and learning algorithm) in order to account for the randomness of the results. Every unique experimental 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 Fig. 8).

4.1 Experimental parameters

The most important parameter of Evolutionary Algorithms [27] is the population size or simply the number of samples ϕ_j 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 stable 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 evolutionary algorithms, such as *MA-ES* and *MA-CES*, 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 NN. In our experiments, every NN has one hidden layer with 64 nodes, and an input size equals the state space of the agent $\|s_{agent}\|$, 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, *CCEA*, *MA-ES*, and *MA-CES* use the same fitness function (Eq. 10), 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. Maintaining the same fitness function across all algorithms was crucial, since we wanted to obtain insights from an equal terms comparison of the algorithms and agent factorizations.

$$Fitness(t) = total\_deliveries(t) \tag{10}$$

In genetic [28] and Evolutionary [27] algorithms, the mutation and crossover steps vary in different applications. For instance, a straightforward way to mutate genes is the addition of noise ϕ_j to the agent policy parameters, θ_t^i as presented in line 5 of Algorithm 5. Of course, there are many alternative options available for mutation when working with

[7] Our source code can be accessed at the following link: https://github.com/StavrosOrf//MultiAgentLearning.

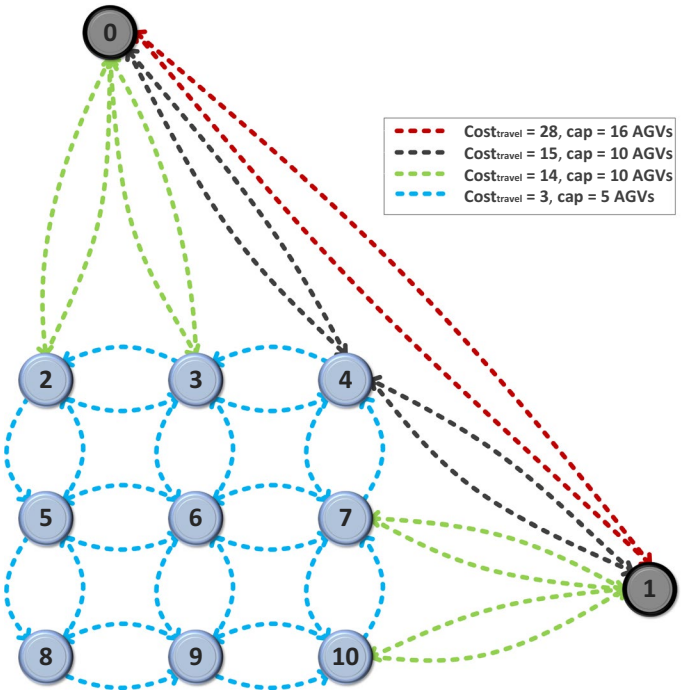

Fig. 8 The warehouse graph used in our experiments [4, 13]. The dashed lines represent the edges with an associated travel cost and capacity (*cap*). Nodes 0 and 1 represent the entrances/exits of the graph, so AGVs start the simulation at these vertexes. Even though the warehouse might appear deceivingly simple, it entails a complex optimization problem since the traffic routing system has to dynamically decide which route (out of many possible ones) the AGVs should take to minimize the travel time while avoiding (costly) congestion

parameters θ_t of neural networks [49, 50]. However, for the mutation step of our algorithms in this study, we simply resolved to add sampled Gaussian noise.

Finally, we summarize the main differences between *MA-ES*, *CCEA*, and *MA-CES*. In detail, *MA-ES* and *MA-CES* are combing the information attained from all the genes (line 7 of Algorithm 5 and line 9 of Algorithm 6), 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. Additionally, even though *MA-ES*, and *MA-CES* have quite similar structures, we found out experimentally that they have substantial differences in terms of performance and convergence speed.

4.2 Evaluating *CCEA*

Initially, we evaluate the performance impact of combining different agent factorization with *CCEA*. As mentioned, this is a multiagent genetic algorithm [28] that generates a large number of genes. By so doing, it aims to find the best solution for an optimization task using mutation and crossover processes based on a fitness function $F()$. We define as "team performance" the amount of the completed deliveries G by all the AGVs in a 200-step simulation.

In Fig. 9, we can see the average training performance across training epochs and many experimental runs, and in Fig. 10, we can see the best team performance across the training process. These figures can help us gain interesting insights into the average and best performance of each agent factorization when using the same algorithm. The left axis depicts the total deliveries G completed after the end of a simulation during a 500-epoch training process.

In the simplest 90 AGVs environment, we can observe that the more decentralized the agent team is, e.g., comparing the *centralized* agent (538 max deliveries, 520 average deliveries) vs. *link* agent definitions (544 max deliveries, 528 average deliveries), the better the results we get. These exhibit fast learning transience, indicating that, when using *CCEA* in the simplest environment (90 AGVs), all agent definitions can coordinate effectively and solve the coordination problem. When examining the impact of adding the *last* or/and *average* travel time to the state resolution (*with * time*[8]), we can observe a trend of reduction in the variance of training performance (as seen in Fig. 9), and a reduction in performance except for the *link* agent teams, where we saw a performance increase, with the *link with both times* agent team performing the best (544 max deliveries, 548 average deliveries). This indicates that even in the simplest environment, our *centralized* and *intersection* teams are limited by the curse of dimensionality,[9] while the *link* teams are limited by the problem's coordination complexity.[10]

In the 120 AGVs environment, we can still observe the trend of the more decentralized agent teams performing better, e.g., *centralized* agent gets 608 average deliveries, 631 max deliveries, while the *link* agent them gets 676 max deliveries, 657 average deliveries. Furthermore, the addition of extra state information, i.e., using *with * time* agent definitions, still negatively impacts the performance of the *centralized* and *intersection* agents teams. Even more so, for the *with both times* agent definitions, except for again the *Link* agent teams, which instead improves its performance with the addition of extra state information, which indicates that *link* agent teams are still not limited by the curse of dimensionality.

In the 200 AGVs environment, we can observe the first change in the learning trend. The best performing (*without * time*) agent team is the *intersection* (842 max deliveries, 744 average deliveries), and the worst performing one is the *centralized* agent (716 max deliveries, 677 average deliveries), with *link* agent team not being far ahead (723 max deliveries, 661 average deliveries), indicating that the *intersection* agent team can be a good compromise between centralization and decentralization. However, as the state resolution is expanded with AGV travel time (*with * time*), the performance of *centralized* and *intersection* agent teams drops. Meanwhile, *link* team's performance improves significantly (even more than in the simpler environments) with *link with avg time* performing the best (894 max deliveries, 785 average deliveries), indicating that the *link* agent teams are starting to be limited by coordination complexity since they can not coordinate efficiently when AGV tracking information is missing (*without * time*).

Finally, in the most complex environment with 400 AGVs, the *intersection* agent team (849 max deliveries, 759 average deliveries) is the best compared to all the *without time*

[8] We use the "with * time" notation to represent all agent definitions that incorporate AGV travel time in any form.

[9] The *curse of dimensionality* refers to the difficulties that arise when working with high-dimensional data due to the exponential increase in volume and sparsity of data as the number of dimensions increases [51]. In our case, such difficulties arise via adding extra state information, i.e., using *with * time* agent definitions or more centralized agent definitions.

[10] The term *coordination complexity* refers to how difficult the problem is for cooperative agents to work together [52].

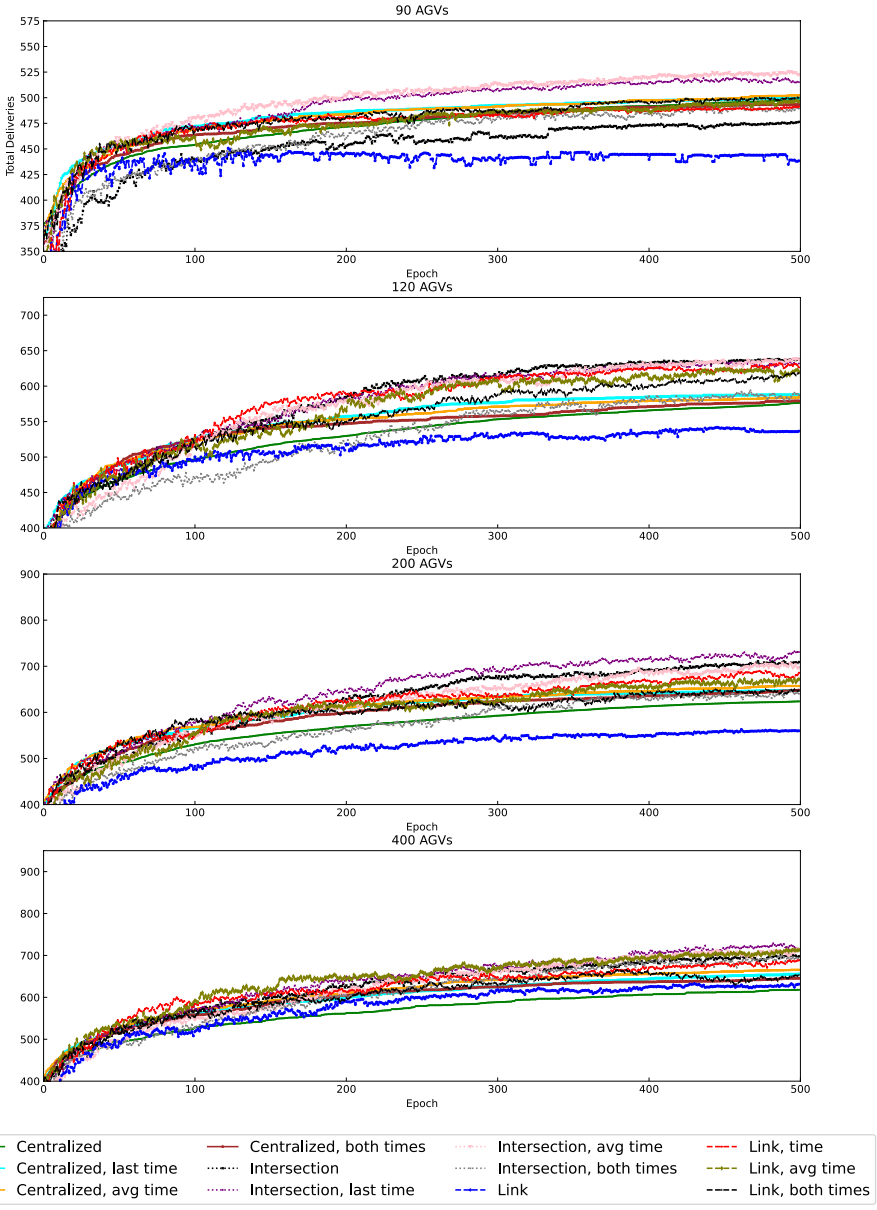

Fig. 9 *Average* team performance across 500 training epochs after 30 experimental runs, using the CCEA learning algorithm. The optimal solution, i.e., the total number of deliveries made, is directly affected by the number of AGVs traversing the warehouse

agent definitions, and the worst performing one is still the *centralized* agent (679 max deliveries, 338 average deliveries). Surprisingly, when time is added (*with * time* agent definitions), all except *intersection both times* showcase performance improvement over *without time* agent definitions. Therefore, this indicates that, in general, coordination complexity

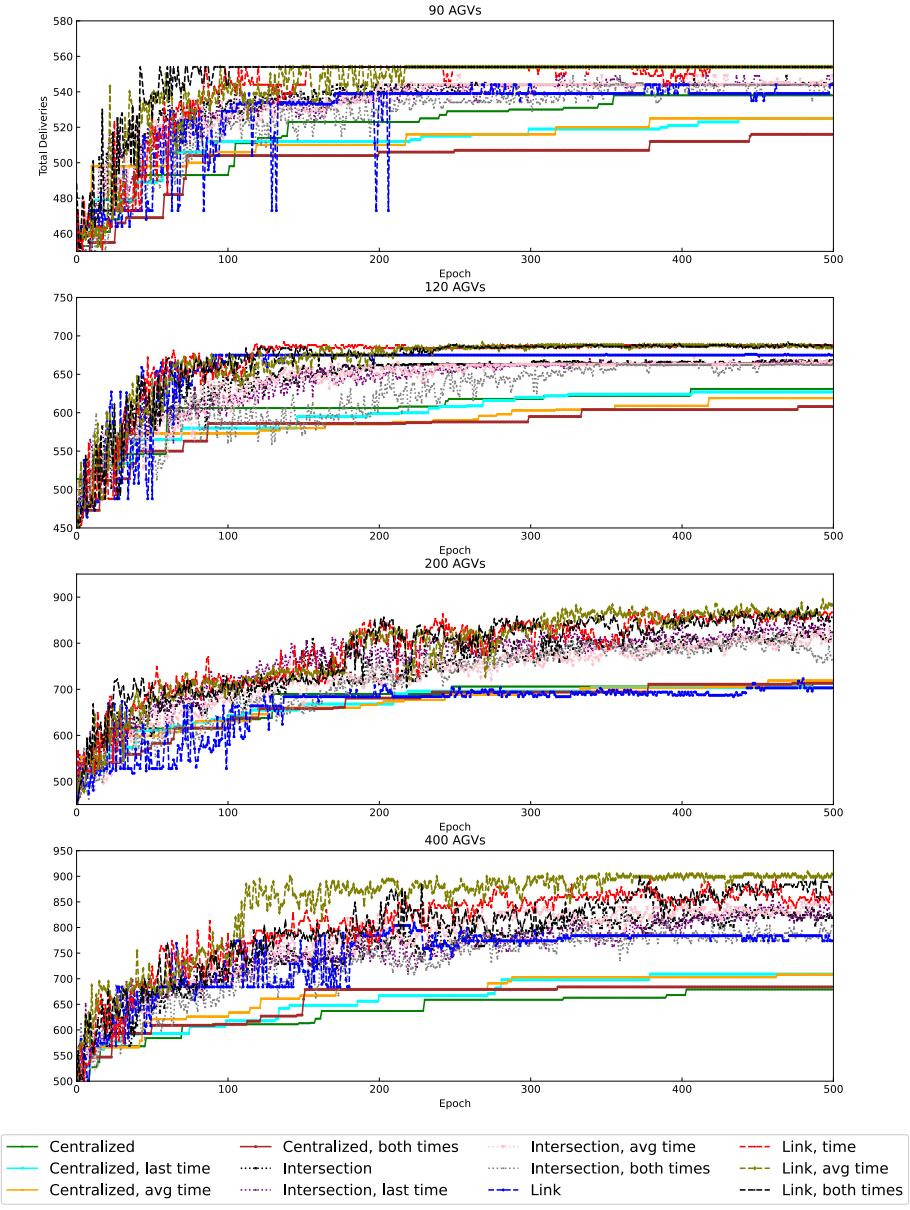

Fig. 10 *Best* team performance across 500 training epochs after 30 experimental runs, using the CCEA learning algorithm

is more prevalent than the curse of dimensionality, in more complex environments. This is because when the state resolution is expanded the dimensionality of the problem increases, while the coordination complexity decreases.

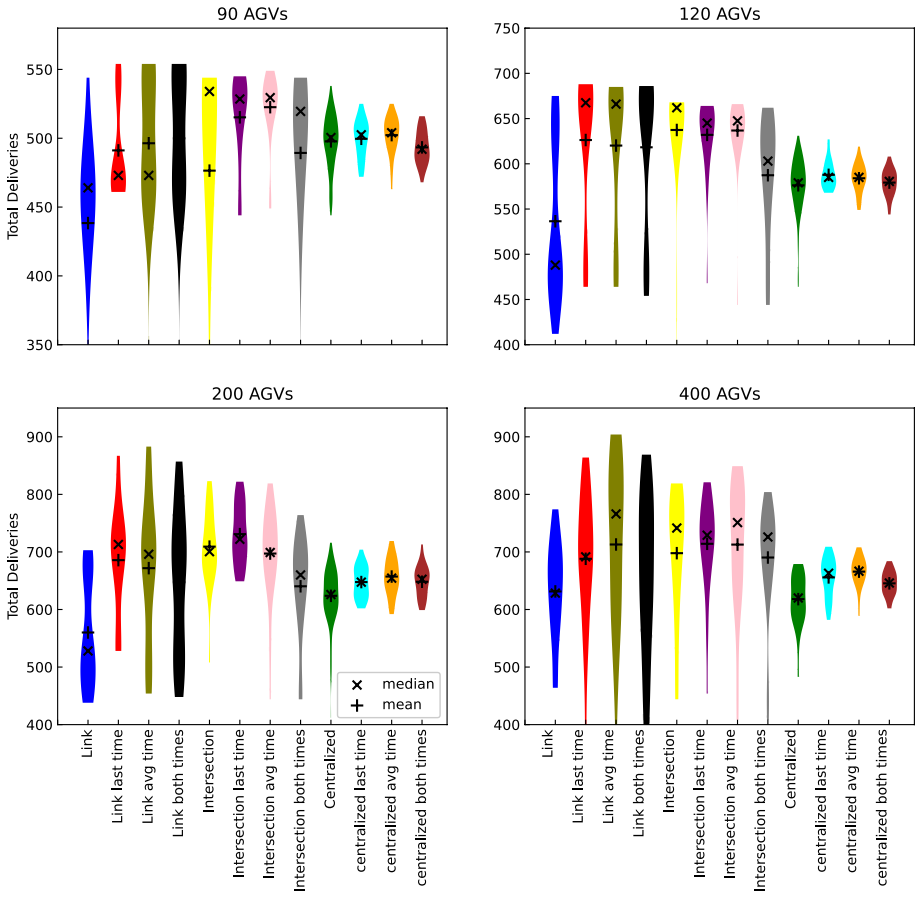

Fig. 11 Violin plots depicting the distribution for every different scenario at the end of the simulations for *CCEA*

Looking at trends encompassing multiple environments, the *centralized* agents do suffer performance losses when expanding the observable space (*with * time*). Specifically, as the environment complexity increases (more AGVs) and the curse of dimensionality becomes more apparent, to the contrary, the *link* agent teams always perform better when *time* is added to observable space. This signifies that, as expected, the *centralized* agents are mostly limited by the curse of dimensionality, while the *link* agent teams are limited by their coordination complexity (as can be observed in Fig. 10). Also, the *centralized* agents exhibit a continuous performance improvement (Fig. 10); this is because of the elitist gene selection policy and the lack of Structural Credit Assignment problem.

Next, we look at Table 3 that compares the convergence time in terms of epochs required to reach 90% of the best solution for each agent definition. In the 90 AGV environment, out of the agent teams that utilize the extra AGV travel time (*with * time*), the * *with avg time* ones, presented a significantly slow learning speed, even though they managed the same overall performance level in the end as the * *with last time* and * *with both times* agent teams. In all the environments, the *centralized with * time* agents converge much faster than the *centralized* (without time) agent, to a much lower

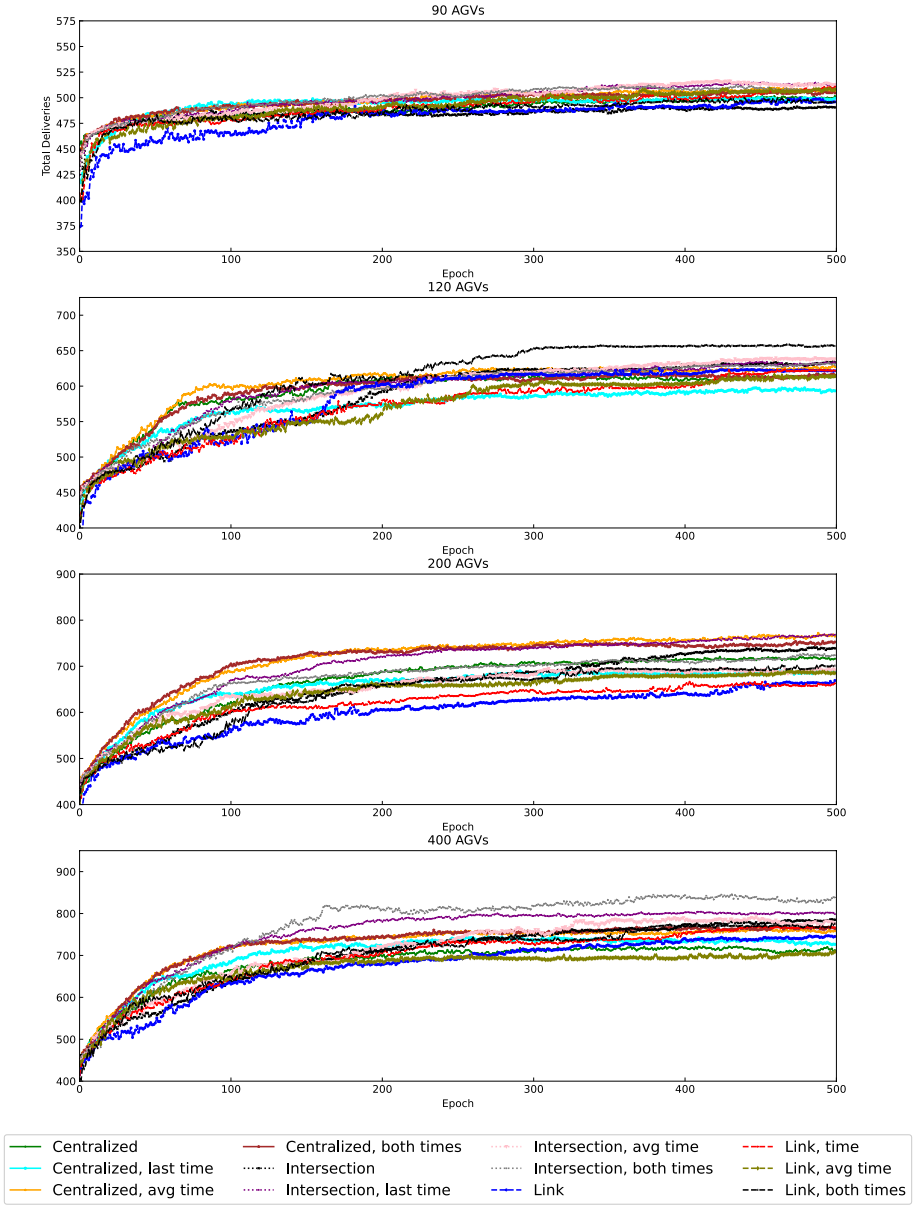

Fig. 12 *Average* team performance across 500 training epochs after 30 experimental runs, using the *MA-ES* learning algorithms

performance level. Moreover, we can see that all agent teams in all the environments using a *link* agent structure learned the fastest, except for *link avg time* in the 90 AGV environment.

Next, using the violin plots Fig. 11, we can see the distribution of performance after training for 500 epochs. In all environments, the *centralized* agents exhibited very little

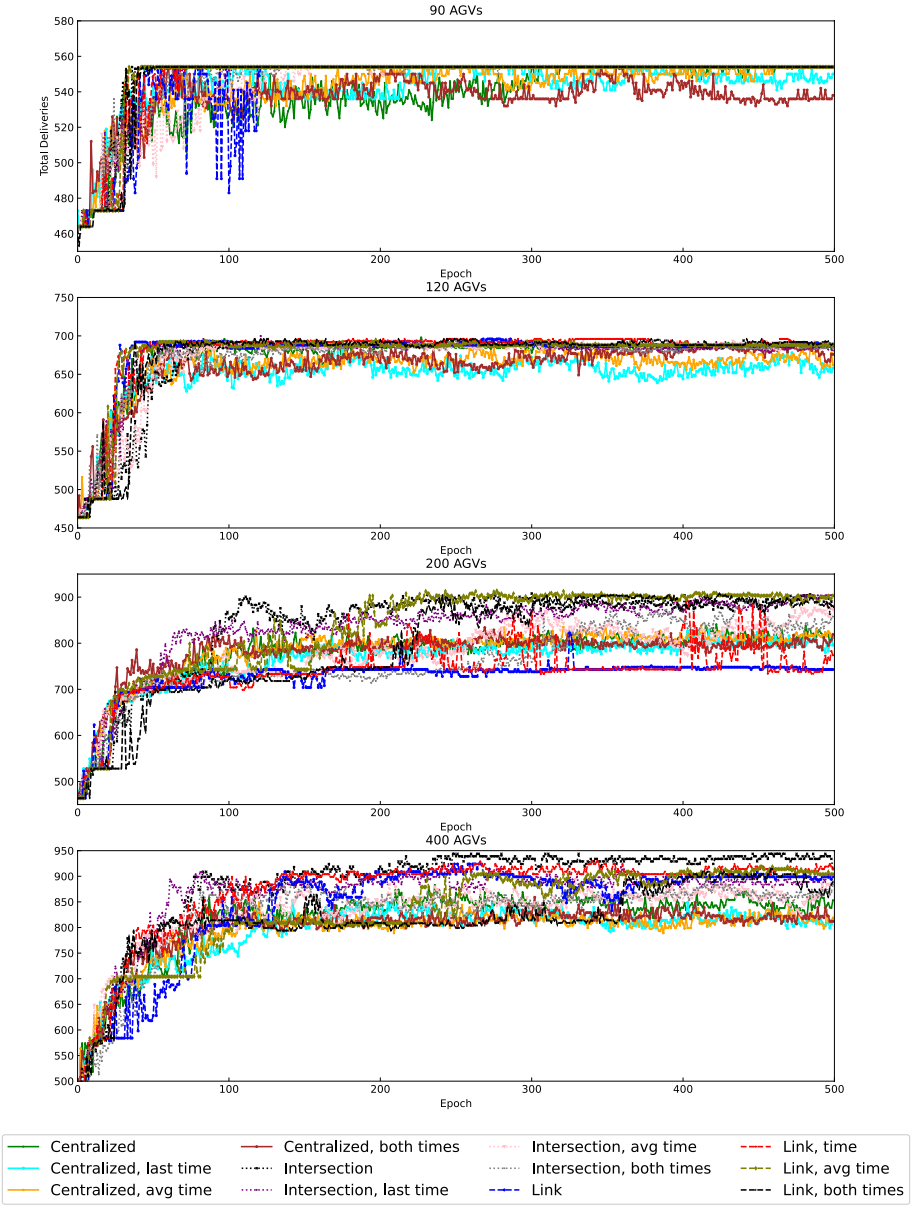

Fig. 13 *Best* team performance across 500 training epochs, using the *MA-ES* learning algorithms

skew (this can be seen, by the overlap of the medians and means). Furthermore, the *centralized with * time* agents exhibit much less variance than the *centralized* agent. Lastly, the *Link* agent team in all environments skew bottom-heavy, whereas all the *link with * time* agent teams skewed top-heavy.

4.3 Evaluating MA-evolutionary strategies

We continue by evaluating the performance of *MA-ES* in this domain. In Fig. 12, we can see the *average team performance* across 500 training epochs.

In Fig. 13, we can observe the best team performances during training. In the simplest environment (90 AGVs), we see that when using *MA-ES*, all agent definitions are able to get the exact same maximum deliveries (554 max deliveries). This happens because the dimensionality of the problem is relatively low, and *MA-ES* can consistently solve this problem with any agent definition.

In the 120 AGVs environment, all agent definitions are able to reach a similar (non-optimal) result, with *intersection with last time* agent team getting the best results (700 max deliveries, 675 average deliveries after training). This indicates that for the 120 AGVs environment, the dimensionality of the problem is still relatively low. Looking at the addition of state resolution (*with * time*), on the *centralized* agent, we can see that the performance scarcely degrades, even though the *centralized* agent performs very well (698 max deliveries, 675 average deliveries), implying that the *centralized with * time* agents are slightly limited by the curse of the dimensionality. Interestingly, the *intersection* and *link* agent teams, are mostly unaffected by the addition of AGV tracking information (*with * time*).

Most importantly, in the 200 AGVs environment, where the complexity of the problem has increased, the *centralized* agent is performing quite poorly, and it gets even worse when time tracking information is added (*centralized with * time*). While the *intersection* (without * time) agent team performs quite well, and when time tracking information is added (*intersection with * time*), the performance slightly degrades (similarly to the *centralized* agents in the 120 AGV environment), because it is limited by the curse of dimensionality. Meanwhile, the most decentralized *Link (without * time)* agent definition performs the worst (822 max deliveries, 726 average deliveries), but greatly benefits from the addition of AGV time-tracking information. In particular, *link with avg time* is performing the best (914 max deliveries, 835 average deliveries) with *intersection (without * time)* being a close second (909 max deliveries, 840 average deliveries). This indicates that the impact of the structural credit assignment problem [11] increases slower than that of the curse of dimensionality.

The performance trend is quite similar for the more complex 400 AGVs environment. Looking at the teams without AGV tracking information, the *intersection* agent team performs the best (944 max deliveries, 889 average deliveries), with the *centralized* agent performing the worst (883 max deliveries, 816 average deliveries). The *centralized* agent and *intersection* agent team suffer a performance penalty for including AGV tracking information (*with * time*), since they are even more restricted by the problem's dimensionality. This indicates that when we get to very complex domains, the curse of dimensionality will be the greatest obstacle to increasing the system's performance. Meanwhile, the *link* agent team only benefits from extra AGV tracking information with *link with last time* benefiting the most (929 max deliveries, 872 average deliveries). Overall, the *intersection* agent definition performs the best (944 max deliveries, 889 average deliveries), by a huge margin, showing that in complex environments, the middle ground in terms of centralized/

[11] The structural credit assignment problem refers to the complexity of assigning agent-specific rewards in multiagent learning problems.

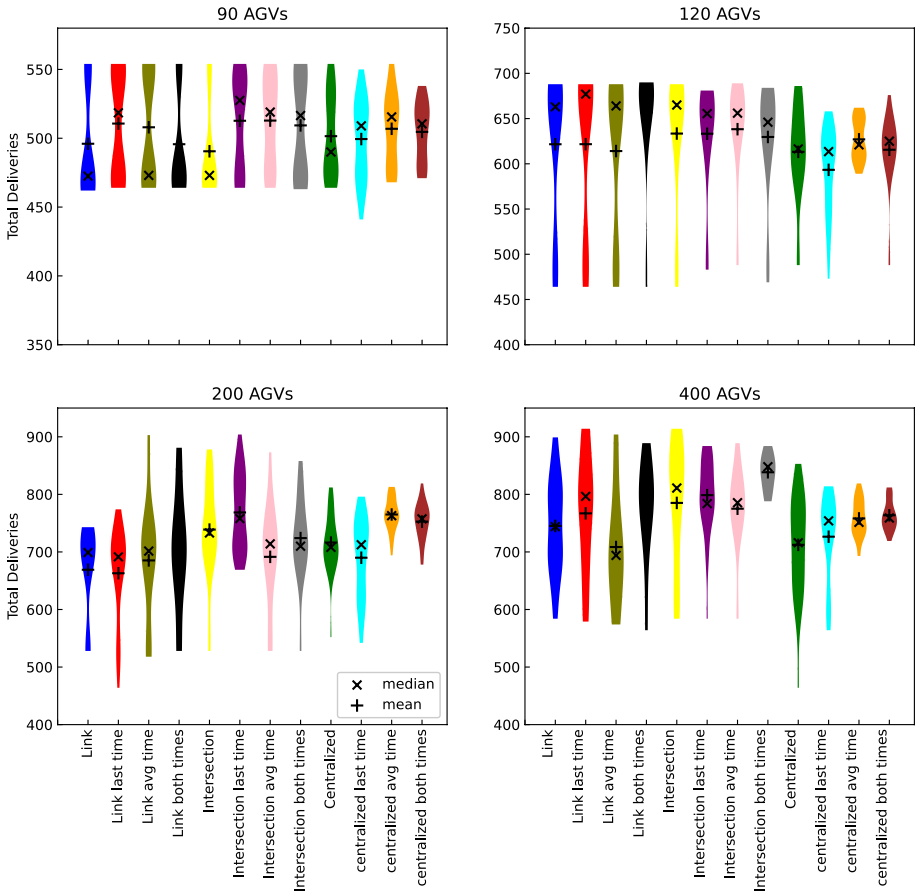

Fig. 14 Violin plots depicting the distribution for every different scenario at the end of the simulations for MA-ES

decentralized agent structures (*intersection* agent team) is a good solution to achieve the best performance.

There are also important insights obtained by observing the convergence time (See Table 3) of the *MA-ES* algorithm. Overall, *centralized* agent teams require significantly fewer epochs to find their best solution. For example, in the 90 AGVs environment, the *link* agents required over 70 epochs to reach 90% of their best solution, while the centralized agents converged much faster, demanding less than 40 epochs on average. Of course, it is impossible to decide on the best algorithm only by observing their convergence time. Therefore, a decision based on both the maximum performance and convergence time is needed.

Figure 14 shows the distributions of the completed deliveries at the final epoch for every case after many experimental runs. In the simple 90 & 120 AGVs environments, we can see that even though all agent teams managed to achieve the optimal outcome, the more decentralized the agent team is (regardless of AGV time tracking observations), the higher variance we get. In the 120 AGVs environment, when adding both AGV tracking information (* *with both times* agent teams), we can observe a reduction in variance. Meanwhile,

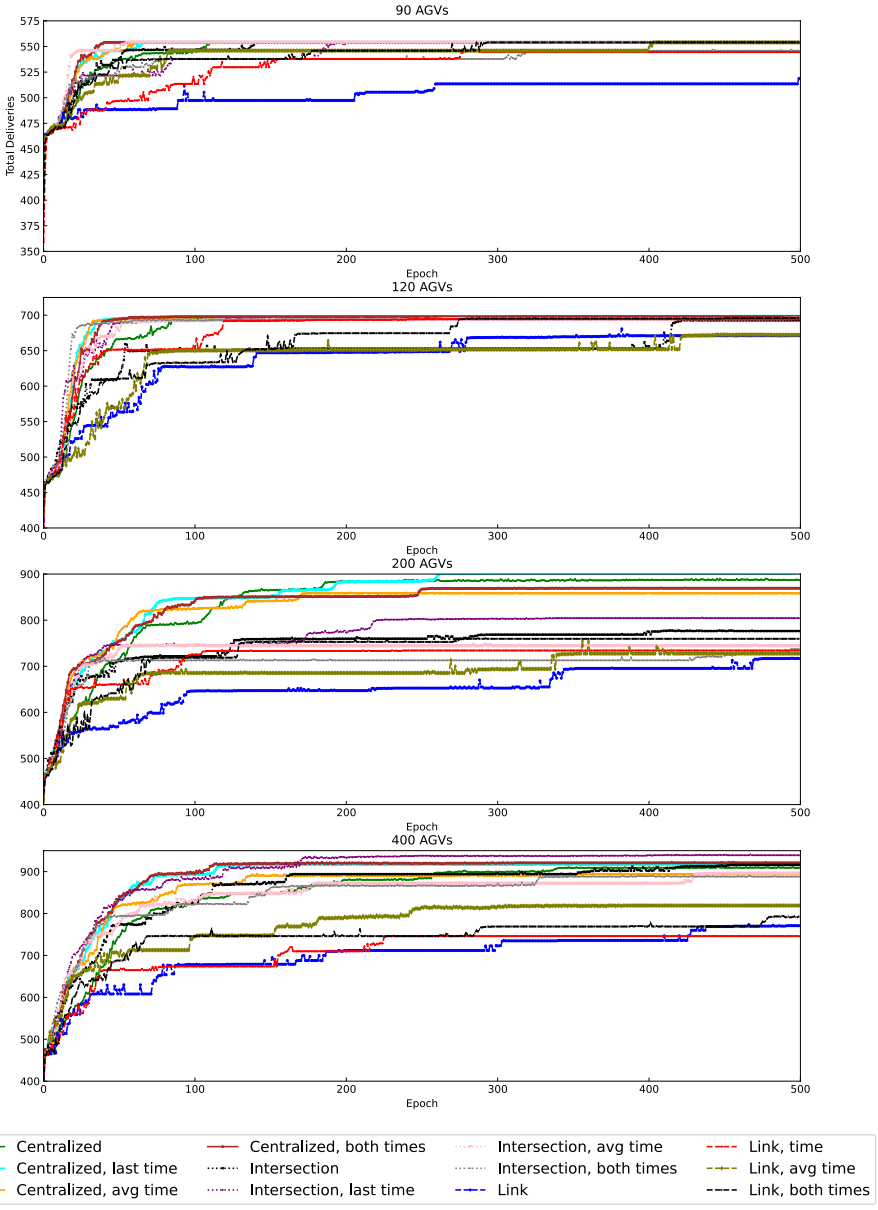

Fig. 15 *Average* team performance across 500 training epochs, using the Multiagent Canonical ES learning algorithm

in the complex 200 & 400 AGV environments, the link agent teams experience an increase in variance with an increase of AGV tracking information. On the contrary, the centralized agents experience a reduction in variances, indicating that the impact of state resolution and agent structure are interlinked. Finally, in the 200 & 400 AGV environments, all agent definitions skew middle (median and average are really close).

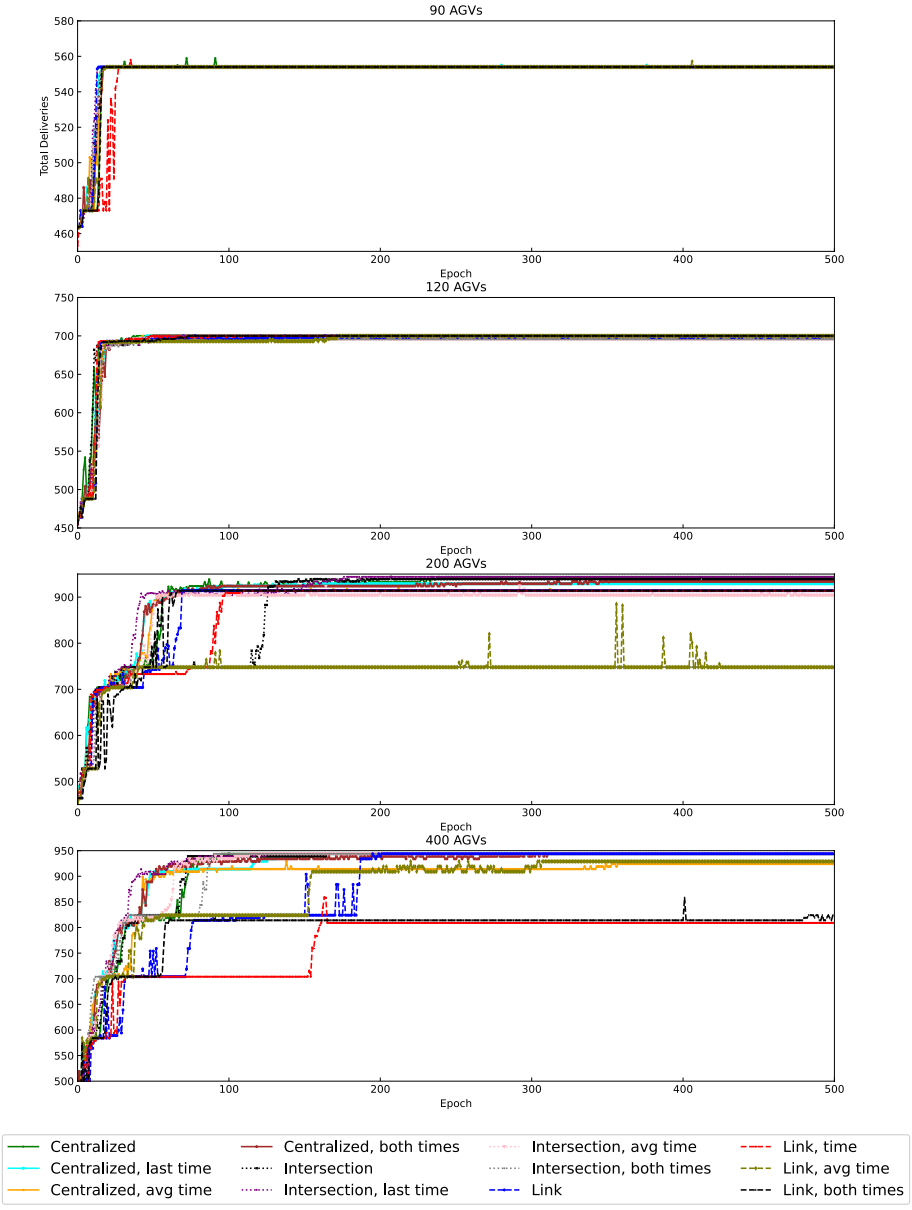

Fig. 16 *Best* team performance across 500 training epochs, using the Multiagent Canonical ES learning algorithm

4.4 Evaluating canonical evolutionary strategies

Lastly, we evaluate the performance impact of various agent definitions when paired with *MA-CES*. Figure 15 shows the average team performance, in terms of total deliveries achieved, after 500 training epochs, while Fig. 16 depicts the best team performance.

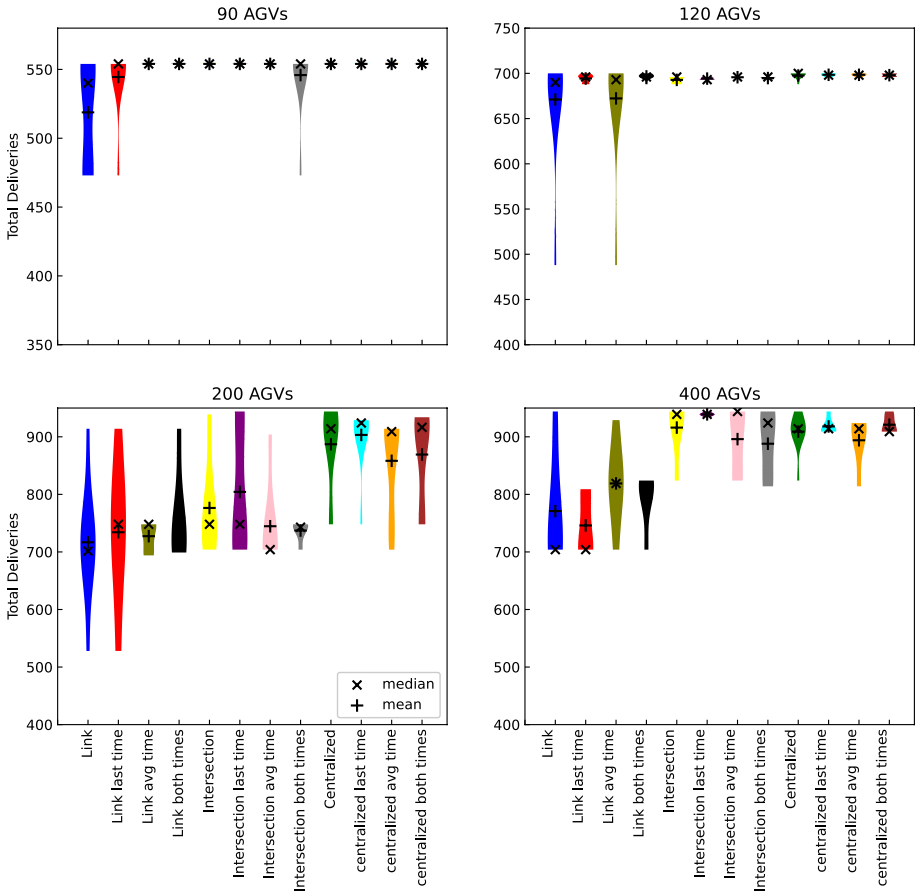

Fig. 17 Violin plots depicting the distribution for every different scenario at the end of the simulations for Canonical ES

Initially, we start by evaluating the performance of *MA-CES* for the 90 and 120 AGVs environments. There *MA-CES* managed to find excellent solutions, with every agent definition (See Fig. 16), with very little variation between agent definitions.

In the more complex environment of 200 AGVs, the impact of different agent definitions on learning performance starts to emerge. Specifically, *MA-CES* managed to reach the experimental maximum of possible deliveries (944 max deliveries) with three different agent definitions, namely *centralized*, *intersection*, and *intersection with last time*. The *centralized* agent experiences performance degradation when AGV tracking info is being incorporated (*centralized with * time*), also, the *intersection with both times* agent team performs by far the worse (748 max deliveries, 740 avg deliveries). Meanwhile, it solved the optimization task sufficiently with other agent definitions, too.

We can observe similar results for the 400 AGVs environment; however, here, *MA-CES* achieved the best results with multiple agent definitions. Interestingly, the agent teams with most decentralized agent structure and AGV tracking info (*link with * time*) do not work as well.

In most cases in all the environments, there was no significant performance difference between the agent definitions that includes AGV tracking information (*with * time*), except for the cases mentioned before.

We have seen that *MA-CES* is an optimization algorithm that is capable of finding great solutions in complex tasks; however, it is not enough to only know if an algorithm can find a very good solution. It is also essential to know how fast it can achieve it and how robust it is. Table 3 demonstrates the convergence time of each algorithm in every case. We can observe that for all environments, *link* agent structures have the highest convergence times than any other agent definition, in most cases. In practice, agent definitions based on the *link* agent structure are more decentralized; thus, it makes sense to require more time to reach the optimal solution.

With the aid of Fig. 17, we can determine the robustness of the *MA-CES* algorithm by observing the violin plots. As mentioned, violin plots represent the distribution of the last team performances after many experimental runs consisting of 500 epochs each. Interestingly, we can observe that *MA-CES* is the most robust algorithm of all three algorithms studied. For 90 and 120 AGVs environments, the violin plots' median and mean values totally overlap on top of the best possible solution, 554 and ≈ 700 deliveries, respectively. Only the *link* agent team demonstrates a slight variance, verifying our claim that *MA-CES* does not operate as well with *link* agent definitions.

The results are more interesting for more complex environments. For the 200 AGVs environment, we can still see that the *centralized* agent definitions are the most consistent, because their violin plots are very top-heavy. In contrast, the violin plots of *link* agent definitions are balanced around their center. This indicates that *MA-CES* paired with *link* agent teams converge to the best solution only half the times it is trained, while when using other agent definitions, it is more likely to converge to a clearly better solution. Similar trends can be observed for the 400 AGVs environment, the only difference here is that the skew of the distributions are significantly smaller.

Overall, *MA-CES* is an algorithm that is capable of finding the very good solutions with most agent definitions, requiring minimum training epochs, while also being extremely robust with *centralized* and *intersection* agent definitions.

4.5 Comparing the learning algorithms

Results in the previous section verified that, indeed, the agents' definitions could make a difference in some multiagent environments. We will now proceed to compare the learning performance of *MA-ES*, *MA-CES*, and *CCEA* algorithm when using different agent definitions.

Table 2 shows the best team performances of the three algorithms in all different scenarios. In the 90 AGVs environment, *MA-ES* and *MA-CES* with all agent definitions managed to converge to solutions that are close to the experimental best policy (559 max deliveries achieved by *centralized* agent definition with *MA-CES*), while *CCEA* only managed to achieve this for the *link with time*. Thus, for this environment, combining *CCEA* with anything other than link-based agents is very suboptimal. Similarly, in the 120 AGVs environment, both the Evolutionary algorithms performed really well with every possible agent definition, while *CCEA* were again incapable of finding the best solutions. In particular, even though *MA-CES* achieved better results with almost every agent definition, with *intersection with last time* agent definition *MA-ES* outperforms *MA-CES*. In the other two environments, namely the 200 AGVs and the 400 AGVs ones, *MA-ES* achieved a much greater

Table 2 Comparison between the maximum deliveries G of *CCEA*, *MA-ES* and *MA-CES*

	90 AGVs			120 AGVs			200 AGVs			400 AGVs		
Algorithm	CCEA	MA-ES	MA-CES	CCEA	MA-ES	MA-CES	CCEA	MA-ES	MA-CES	CCEA	MA-ES	MA-CES
Centr	538	554	**559**	631	698	700	716	841	**944**	679	883	**944**
Centr. last time	525	554	555	627	693	700	704	828	929	709	859	**944**
Centr. avg. time	525	554	554	619	689	700	719	838	919	708	859	929
Centr. both times	516	554	554	608	688	700	713	834	934	684	839	**944**
Inter	549	554	554	669	695	698	842	909	**944**	849	**944**	**944**
Inter. last time	549	554	554	673	700	696	854	904	**944**	864	924	**944**
Inter. avg. time	549	554	554	672	695	698	832	881	924	859	894	**944**
Inter. both times	549	554	554	665	696	696	823	901	748	819	894	**944**
Link	544	554	554	676	696	700	723	822	914	809	924	**944**
Link last time	554	554	558	692	696	**701**	874	893	914	899	929	859
Link avg. time	554	554	557	691	695	700	894	914	885	908	919	929
Link both times	554	554	554	692	696	700	879	908	914	899	914	858

Bold numbers correspond to the maximum deliveries attained by each combination of algorithm and agent definition for every AGV environment

Table 3 Comparison of the convergence time, in terms of the average number of epochs required to reach the 90% of the best solution, of every combination of learning algorithm *CCEA*, *MA-ES* and *MA-CES* and agent definitions

	90 AGVs			120 AGVs			200 AGVs			400 AGVs		
Algorithm	CCEA	MA-ES	MA-CES	CCEA	MA-ES	MA-CES	CCEA	MA-ES	MA-CES	CCEA	MA-ES	MA-CES
Centr	91.7	37.3	30.3	162.6	74.7	32.4	169.3	122.7	84.9	174.6	94.4	119.6
Centr. last time	49.9	17.5	18.6	106.8	92.6	21.6	136.1	99.0	99.9	157.6	99.7	**62.6**
Centr. avg. time	64.4	21.4	20.9	114.7	83.0	20.8	159.6	132.9	61.4	159.5	85.4	74.6
Centr. both times	60.1	23.5	18.5	116.2	78.6	23.0	147.0	96.7	73.3	146.7	81.6	63.2
Inter	59.2	32.5	27.1	124.0	145.4	105.2	178.1	189.5	44.8	161.6	168.6	147.2
Inter. last time	70.6	47.7	55.0	123.6	115.0	22.2	186.4	150.3	63.6	220.6	121.2	80.0
Inter. avg. time	76.6	61.5	**13.3**	117.7	129.0	24.0	206.9	117.5	23.2	201.1	175.5	144.8
Inter. both times	75.4	36.7	24.0	150.7	113.8	**17.0**	180.3	126.5	**19.2**	185.2	116.6	163.0
Link	**13.3**	88.0	69.6	67.4	116.3	107.8	110.5	127.5	168.6	121.1	179.7	186.6
Link last time	42.4	129.8	80.7	110.0	114.2	38.6	114.3	131.1	27.6	122.3	154.0	93.6
Link avg. time	80.4	70.8	71.3	111.4	153.8	76.9	119.3	91.1	151.6	138.4	108.7	124.4
Link both times	44.4	74.4	45.8	90.4	103.7	74.4	134.7	157.3	57.0	141.6	138.2	141.2

Bold numbers correspond to the lowest average number of epochs attained by each combination for every environment

number of completed deliveries compared to that of the *CCEA*, while *MA-CES* managed to get better results compared to the two other algorithms with most agent definitions (but not all agent definitions). Note that *MA-CES* consistently reaches 944 total deliveries [12] with most agent definitions.

More specifically, we can see that in more complex environments (those with higher AGVs count), the improvement gap of the *Evolutionary Strategies* algorithms compared to *CCEA* is much larger than that of simpler environments (lower AGVs count). In the 200 AGVs environment, all MA-ES-generated policies resulted in 20 to around 120 more deliveries compared to *CCEA*. Meanwhile, *MA-CES* generated better results than *MA-ES* (up to 40 more deliveries) in most cases (but worse in others). This shows that this simulated stochastic gradient ascent algorithms *MA-ES* and *MA-CES* works better than a selection-based genetic algorithm (*CCEA*). Furthermore, it is clear that a weighted simulated stochastic ascent algorithm, such as *Canonical Evolutionary Strategies*, is the best fit to solve the complex autonomous warehouse optimization problem.

Furthermore, it is essential to question whether it was beneficial to have more complex state spaces, i.e. utilizing *with both times* agent definitions. For the 90 and 120 AGVs environments, incorporating *with both times* information was not evidently helpful, except for *MA-ES intersection both times*, because the optimization task was easier to solve with any algorithm combined with any agent definition. Also, for the more complex 200 AGVs environment, the extra state-space information did not help improve the results of less decentralized agent definitions, especially for *MA-CES intersection with both times* agent definition (748 max deliveries); however, it was very useful for the *link* agent definitions. The same trend is preserved for the 400 AGVs environment. However, notice that *MA-CES centralized both times* got equally good results with the other agent definitions on *MA-CES*. Overall, *with both times* state information did not help any algorithm achieve the best results, with the single exception being *link with both times* in the 200 AGVs environment and *MA-ES intersection both times* in 120 AGVs environment. However, it should be mentioned that using additional state information (*with time* agent definitions) did not limit the optimization capabilities of the algorithms, as depicted in simpler environments (90 and 120 AGVs) Also, in more complex environments, it was, in most cases, better (in achieving a larger number of final deliveries) to include extra state-space information.

Another fruitful comparison can be conducted between agent definitions utilizing *with last time* and *with average time*. It appears to be no difference in the 90 AGVs environment, because, as mentioned earlier, the optimization problem is simple. However, minor differences can be observed in the 120 AGVs environments. In detail, each non-centralized agent definition utilizing an AGV travel time, seems to perform better, by a few deliveries, with at least one different learning algorithm. More noticeable discrepancies can be observed for the 200 and 400 AGVs environments. There, each algorithm matches best with different agent definitions, both in terms of agent team structure and agent state space.

Looking in general at the impact of state resolution (incorporating time) on "maximum deliveries" performance, we can observe that in all cases, it is primarily insignificant, except for the complex 200 AGVs & 400 AGVs environments of *link with * time* agents, indicating that the impact of state resolution is most prevalent in complex

[12] For this specific warehouse management domain, we consider 944 total deliveries to be the maximum experimental solution in 200 & 400 AGVs environments, since no other algorithms paired with agent definition managed to achieve better results. In addition, the very low variance observed supports the claim that these are most likely the optimal values, experimentally identified.

environments with high decentralization of agent structures. However, notice that the *MA-CES* algorithm on the 400 AGVs environment seems to be indifferent regarding the best agent definition since, it obtains the best results with multiple agent definitions.

Lastly, looking at overall trends, we can observe that the centralized agents on evolutionary strategy algorithms we tested (*MA-ES* & *MA-CES*) do not benefit from the expansion of observable space (*centralized with * time*), while *CCEA*'s *centralized* agents do benefit from it, in the complex environments (200 AGVs & 400 AGVs).

Following this, we can further investigate the differences between the algorithms by comparing the way they learn over the course of 500 training epochs. Table 3 shows the average convergence time of every combination of learning algorithm and agent definition. We define, as convergence time, the training epochs required by an algorithm to reach over 90% of its best performance. Therefore, this metric clearly illustrates how quickly an algorithm can be trained. Of course, faster convergence times do not guarantee better solutions. Convergence time (Table 3) and best-team performance (Table 2) are two metrics that should be assessed in conjunction with each other.

In general, it is clear that *CCEA* is the algorithm that requires the most epochs to find its solutions, especially for *centralized* agent definitions. Additionally, although we expected that the more state space information an agent definition incorporates, the greater the convergence time required, we did not observe any apparent correlation between convergence time and agent definitions incorporating *with both times*. Except for *MA-CES* for all agent definitions, and *CCEA* with *centralized*, which results in faster convergence times when *with both times* is incorporated. Something that we can observe and verify is that the most complex the environment, in general, the more epochs are required to find the best solution. Therefore, this indicates that for more complex environments, all algorithms struggle to find the best solution; thus, they require extra training time. Meanwhile, when algorithms train for more epochs, it is possible to find a better solution; hence, higher convergence speeds can also lead to better solutions, if the algorithms are combined with appropriate agent definitions.

Another interesting observation is that the *link* agent definitions have, in general, lower convergence time when paired with *CCEA*, compared to the other agent definitions of *CCEA*, especially for more complex environments (high AGV counts). However, the *link* agent definition, in general, requires more time when paired with *Canonical Evolutionary Algorithm*. This evidently supports our initial claim [13] that the pairing of a learning algorithm and an agent definition is significantly more impacting than just the agent definition, as it was previously suggested by [12].

Arguably, both *Evolutionary Strategies* algorithms that were demonstrated in this study, and especially *MA-CES*, are capable of finding the very good solutions (See Table 2), while also achieving that in lower training times than *genetic algorithms*. Lastly, while *MA-CES* outperformed *MA-ES* with most agent definitions, it is essential to highlight that for some other agent definitions, this was not always occurring (e.g. 120AGVs *intersection with last time* and 200 AGVs *intersection with both times*).

In most cases, *MA-CES* outperforms *MA-ES* because its state-action space contains plateau regions (that have 0 gradients w.r.t. total deliveries), in which *MA-ES* tend to get stuck, since most of the exploration steps will have no performance difference and the simulated gradient's expected value would therefore be close to 0 (for some regions $E[\nabla F(\theta_t + \sigma\phi)] \approx 0$), while MA-CES's weighted gradients do not (for the same regions, $E[w\nabla F(\theta_t + \sigma\phi)] \neq 0$), this is why in particular in the 90 AGV environment *MA-ES*'s performance max is 559 total deliveries.

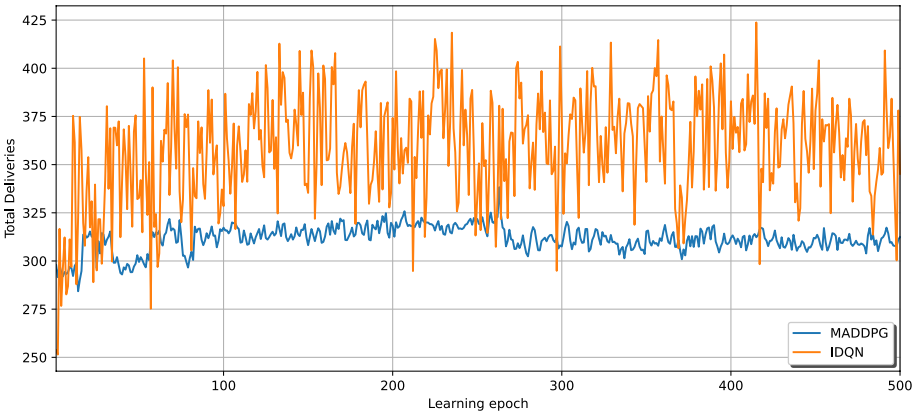

Fig. 18 Total deliveries G for the 120 AGVs environment using MADDPG and I-DQN algorithms after 500 training epochs: Best results across all 30 runs involving experimentation with three different reward functions. The run depicted is using the *RF2* function—i.e., the one that rewards progress towards the AGVs' destinations

4.6 Testing reinforcement learning methods

Evolutionary Strategies and Canonical Evolutionary Strategies are evolutionary algorithms that simulate a stochastic gradient ascent; inspired by this, we wanted to study how policy gradient algorithms and other prominent MARL algorithms perform in this domain.

At first, we implemented the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [19] algorithm, which has proven to be capable of solving complex RL tasks in a wide variety of domains [53]. We also implemented the decentralized I-DQN algorithm, where each agent has its own independent training neural network. However, neither algorithm managed to train all their independent or centralized agents, due to the unique characteristics of the warehouse traffic management domain. Generally, most reinforcement learning algorithms, like MADDPG and I-DQN, require carefully designed reward functions that will train the agent policies. As such, we experimented with three different reward functions:

- *RF1:* a function that incentivizes the AGVs to move on the shortest available path.
- *RF2:* a function that estimates progress towards the AGVs' destinations via counting the number of AGVs that are eligible to move towards their destination.
- *RF3:* a function that assesses deliveries' progress within specific time windows.

Our tests showed that there was no obvious reward function resulting in the effective training of the agents. More specifically, we ran at least 10 learning simulations (over 500 training epochs each) for each reward-algorithm pairing, but none of these managed to converge (i.e., no learning was demonstrated).

Figure 18 illustrates the best results obtained for those algorithms across all reward functions tested. Specifically, the figure depicts the results of using reward function *RF2*—i.e., the one that provides as a reward the number of AGVs moving toward their goal. It is obvious even in this "best" run that there is no convergence, while performance reaches at

most 425 deliveries which is 60% of the value found experimentally to be optimal in this domain (i.e., 701 deliveries, cf. Table 2).

In particular, we believe that the main difficulty of 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 [54]. 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 "credit assignment problem" [55] 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, hyperparameters, and reward functions tested. Even 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. In other words, had we taken a different agent structure as a constant (e.g., intersection agents) I-DQN would have been unimplementable. This underscores the value of *MA-ES*, *MA-CES* and *CCEA* as alternative learning methods of choice, which can be applied in many complex domains that suffer from the problems mentioned above [56].

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 and where new possibilities are enabled. For example, in our domain, if the *link with time* agent definition were taken as a constant, then *CCEA* would have been in the same performance ballpark as *MA-ES*—and after some further fine-tuning, it might have delivered slightly better results. On the other hand, if the selected agent definition were the *intersection with time*, *MA-ES* would have performed significantly better. Another example is that if we were trying to find the best solution for the 200 AGVs environment, and we had selected *intersection both times* agent definition, we would have concluded that *MA-ES* is able to provide a good solution, and we would have missed the opportunity to exploit the high-performance capabilities of *MA-CES*.

By contrast, in this work we tried to simultaneously assess the performance impact of agent factorization in conjunction with different learning algorithms, in a multiagent coordination setting of real-world interest; and thus discover the source of performance quality for the various agent factorization-MA learning algorithms combinations.

To this end, we put forward and evaluated the multiagent evolutionary algorithms *MA-ES* and *MA-CES*. We also introduced and tested 12 overall (6 new) agent definitions for the warehouse AGV framework. We demonstrated that both multiagent Evolutionary Strategies algorithms we put forward are an excellent fit for warehouse traffic management framework, and they outperform *CCEA* [4] in most cases.

Indeed, we confirmed that the impact of different agent definitions differs based on the learning algorithm used to train the agents. In particular, we showed that using 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 more effectively by using evolutionary algorithms, especially when there are delayed rewards from the environment. Arguably, both *Evolutionary Strategies* algorithms that were demonstrated in this study, and especially *MA-CES*, are capable of finding the best solutions, while also achieving that in lower training times than *CCEA*, which is a genetic algorithm [28].

Difference rewards have the potential to enhance the collective policy learned by multiple reinforcement learning agents operating concurrently within a shared environment. Therefore, the credit assignment problem could be eased with the use of difference rewards [57]. Furthermore, there are numerous opportunities for future research to explore the combination of agent factorization with (multiagent) learning, not only in warehouse traffic management but also in other important real-world domains of interest, such as robotics [58], distributed control [59], telecommunications [60], and economics [61]. A promising avenue of investigation entails assessing a diverse range of agent definitions, encompassing hierarchical structures and factorizations employing distinct communication channels across diverse domains and frameworks, such as MuJoCo based environments [62], and Brax [63]. Additionally, an array of alternative learning algorithms, such as reinforcement learning algorithms, can be utilized to pinpoint the source of the impact of diverse agent definitions. In particular, devising novel multiagent learning algorithms that can exploit specific agent factorizations, is an interesting future work topic that should be thoroughly investigated. Lastly, competitive domains like pricing strategies in markets [64], and potentially mixed domains represent a range of prospective application settings.

Additional experimental data

For the sake of completeness, in this appendix, we are presenting detailed figures of our extensive evaluation. Namely, Fig. 19 depicts an average team performance comparison for every possible combination of agent definition and learning algorithm after many experimental runs. Meanwhile, Fig. 20 presents the best team performance comparison.

Fig. 19 Average team performance comparison between Evolutionary Strategies (*MA-ES*), Canonical Evolution Strategies (*MA-CES*), and the Cooperative Coevolutionary Algorithm (*CCEA*). Full lines are for *CCEA*, dashed lines are for *MA-ES*, and dotted lines are for *MA-CES*

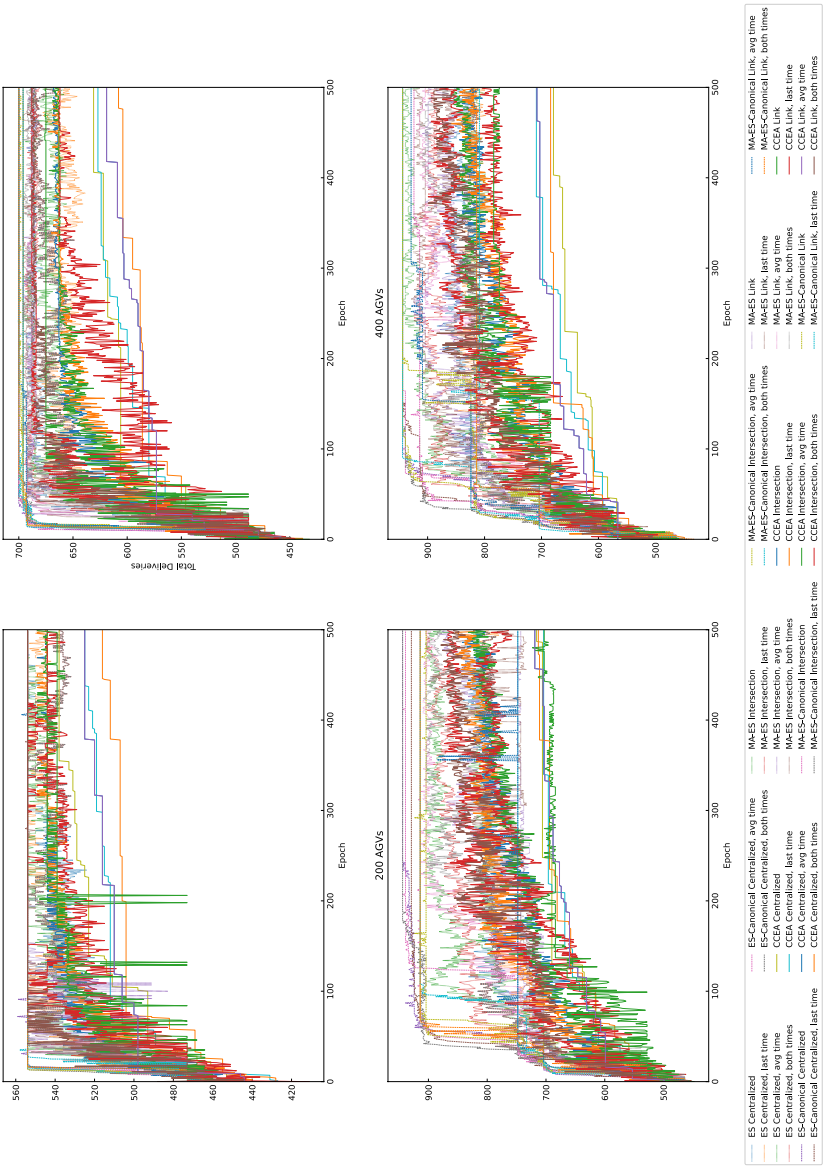

Fig. 20 Best team performance comparison between Evolutionary Strategies (MA-ES), Canonical Evolution Strategies (MA-CES), and the Cooperative Coevolutionary Algorithm (CCEA). Full lines are for *CCEA*, dashed lines are for *MA-ES*, and dotted lines are for *MA-CES*

Acknowledgements The research described in this paper was carried out within the framework of the National Recovery and Resilience Plan Greece 2.0, funded by the European Union - NextGenerationEU (Implementation Body: HFRI. Project name: DEEP-REBAYES. HFRI Project Number 15430).

Author Contributions Conceptualization, A.K., S.O., G.C.; Methodology, A.K. and S.O.; Writing original draft, A.K. and S.O.; Writing review & editing, A.K., S.O., G.C. All authors reviewed the manuscript.

Funding European Union - NextGenerationEU - National Recovery and Resilience Plan Greece 2.0. Implementation Body: HFRI, Grant number: HFRI Project Number 15430 (project name: "DEEP-REBAYES").

Declarations

Conflict of interest The authors declare no conflict of interest.

References

1. Dyke, D. V. (1994). Applications of distributed artificial intelligence in industry. In S. Tahm (Ed.), *Foundations of distributed artificial intelligence.* Industrial Technology Institute.
2. Ye, D., Zhang, M., & Yang, Y. (2015). A multi-agent framework for packet routing in wireless sensor networks. *Sensors (Basel, Switzerland), 15*, 10026–10047.
3. Hassan, M. H., Jubair, M. A., Mostafa, S. A., Kamaludin, H., Mustapha, A., Fudzee, M. F. M., & Mahdin, H. (2020). A general framework of genetic multi-agent routing protocol for improving the performance of manet environment. *IAES International Journal of Artificial Intelligence, 9*, 310–316.
4. Chung, J. J., Rebhuhn, C., Yates, C., Hollinger, G. A., & Tumer, K. (2019). A multiagent framework for learning dynamic traffic management strategies. *Autonomous Robots, 43*, 1375–1391.
5. Ghosh, S., Laguna, S., Lim, S. H., Wynter, L., & Poonawala, H. A. (2020). A deep ensemble multi-agent reinforcement learning approach for air traffic control. arXiv:abs/2004.01387
6. Ramchurn, S., Vytelingum, P., Rogers, A., & Jennings, N. (2012). Putting the 'smarts' into the smart grid: A grand challenge for artificial intelligence. *Communications of the ACM, 55*, 86–97.
7. Murugesan, S., Jiang, Z., Risbeck, M. J., Amores, J., Zhang, C., Ramamurti, V., Drees, K. H., & Lee, Y. M. (2020). Less is more: Simplified state-action space for deep reinforcement learning based hvac control. In *Proceedings of the 1st international workshop on reinforcement learning for energy management in buildings & cities* (pp. 20–23). New York, NY, USA: ACM.
8. Son, K., Kim, D., Kang, W. J., Hostallero, D. E., & Yi, Y. (2019). Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. arXiv:abs/1905.05408
9. Peng, B., Rashid, T., Witt, C. S., Kamienny, P. -A., Torr, P., Wendelin, B., & Whiteson, S. (2021). Facmac: Factored multi-agent centralised policy gradients. In *NeurIPS*.
10. Li, Z., Zhao, W., Wu, L., & Pajarinen, J. (2024). Agentmixer: Multi-agent correlated policy factorization. arXiv:abs/2401.08728
11. Liu, S. (2023). Research of multi-agent deep reinforcement learning based on value factorization. *Highlights in Science, Engineering and Technology, 39*, 848–854.
12. Chung, J. J., Miklic, D., Sabattini, L., Tumer, K., & Siegwart, R. (2020). The impact of agent definitions and interactions on multiagent learning for coordination in traffic management domains. *Autonomous Agents and Multi-agent Systems, 34*, 1–27.
13. Kallinteris, A., Orfanoudakis, S., & Chalkiadakis, G. (2022). The performance impact of combining agent factorization with different learning algorithms for multiagent coordination. In *Proceedings of the 12th Hellenic conference on artificial intelligence, SETN '22*. Association for Computing Machinery.
14. Chen, F.-Y., Wang, H., Xie, Y., & Qi, C. (2016). An ACO-based online routing method for multiple order pickers with congestion consideration in warehouse. *Journal of Intelligent Manufacturing, 27*, 389–408.
15. Potter, M. A., & De Jong, K. A. (1994). A cooperative coevolutionary approach to function optimization. In Y. Davidor, H.-P. Schwefel, & R. Männer (Eds.), *Parallel problem solving from nature — PPSN III* (pp. 249–257). Springer.
16. Salimans, T., Ho, J., Chen, X., & Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. arXiv:abs/1703.03864

17. Chrabaszcz, P., Loshchilov, I., & Hutter, F. (2018). Back to basics: Benchmarking canonical evolution strategies for playing atari. In *Proceedings of the 27th international joint conference on artificial intelligence, IJCAI'18* (pp. 1419–1426). AAAI Press.
18. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature, 518*(7540), 529–533, 2.
19. Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., & Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. *NIPS 2017*. NIPS.
20. Digani, V., Hsieh, M. A., Sabattini, L., & Secchi, C. (2019). Coordination of multiple AGVS: a quadratic optimization method. *Autonomous Robots, 43*, 539–555.
21. Qi, M., Li, X., Yan, X., & Zhang, C. (2018). On the evaluation of AGVS-based warehouse operation performance. *Simulation Modelling Practice and Theory, 87*, 379–394.
22. Karakatič, S., & Podgorelec, V. (2015). A survey of genetic algorithms for solving multi depot vehicle routing problem. *Applied Soft Computing, 27*, 519–532.
23. Mallidis, I., Dekker, R., & Vlachos, D. (2012). The impact of greening on supply chain design and cost: a case for a developing region. *Journal of Transport Geography, 22*, 118–128.
24. Wurman, R. P., D'Andrea, R., & Mountz, M. (2008). Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine, 29*(1), 9.
25. Agogino, A., & Tumer, K. (2004). Efficient evaluation functions for multi-rover systems. In K. Deb (Ed.), *Genetic and evolutionary computation–GECCO 2004* (pp. 1–11). Springer.
26. Boyd, S., & Vandenberghe, L. (2004). *Convex optimization*. Cambridge University Press.
27. Bäck, T., & Schwefel, H.-P. (1993). An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation, 1*(1), 1–23, 03.
28. Holland, J. H. (1975). Adaptation in natural and artificial systems. *An introductory analysis with applications to biology, control, and artificial intelligence* (pp. 89–120). Michigan Press.
29. Sutton, R., McAllester, D. A., Singh, S., & Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. In *NIPS*.
30. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning, second edition: An introduction. Adaptive computation and machine learning series*. MIT Press.
31. Rudolph, G. (1997). *Convergence properties of evolutionary algorithms*. Verlag Dr Kovač.
32. Haber, E., & Ruthotto, L. (2017). Stable architectures for deep neural networks. *Inverse Problems, 34*, 014004.
33. Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2016). Continuous control with deep reinforcement learning. In *4th international conference on learning representations, ICLR 2016, San Juan, Puerto Rico, May 2–4*.
34. Sigaud, O. (2023). Combining evolution and deep reinforcement learning for policy search: A survey. *ACM Transactions on Evolutionary Learning and Optimization, 3*(3), 1–20.
35. Pourchot, A., & Sigaud, O. (2019). Cem-rl: Combining evolutionary and gradient-based methods for policy search.
36. Fujimoto, S., Hoof, H., & Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *International conference on machine learning* (pp. 1587–1596). PMLR.
37. Boutilier, C., Dean, T., & Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research, 11*(1), 1–94.
38. Chung, J. J., Chow, S., & Tumer, K. (2018). When less is more: Reducing agent noise with probabilistically learning agents. In *Proceedings of the 17th international conference on autonomous agents and multiagent systems, AAMAS '18* (pp. 1900–1902). Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.
39. Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V. F., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., & Graepel, T. (2017). Value-decomposition networks for cooperative multi-agent learning. In *Adaptive agents and multi-agent systems*.
40. Rashid, T., Samvelyan, M., Witt, C. S., Farquhar, G., Foerster, J. N., & Whiteson, S. (2018). Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. arXiv, arXiv:abs/1803.11485
41. Lyu, X., Xiao, Y., Daley, B., & Amato, C. (2021). Contrasting centralized and decentralized critics in multi-agent reinforcement learning. arXiv CoRR, arXiv:abs/2102.04402
42. Findik, Y., Robinette, P., Jerath, K., & Ahmadzadeh, S. R. (2023). Impact of relational networks in multi-agent learning: A value-based factorization view. In *2023 62nd IEEE conference on decision and control (CDC)* (pp. 4447–4454).

43. Lee, H., & Jeong, J. (2021). Mobile robot path optimization technique based on reinforcement learning algorithm in warehouse environment. *Applied Sciences, 11*, 1209.
44. Kordos, M., Boryczko, J., Blachnik, M., & Golak, S. (2020). Optimization of warehouse operations with genetic algorithms. *Applied Sciences, 10*(14), 4817.
45. Bao, L. G., Dang, T. G., & Anh, N. D. (2019). Storage assignment policy and route planning of agvs in warehouse optimization. In *2019 International conference on system science and engineering (ICSSE)* (pp. 599–604).
46. Markowski, T., & Bilski, P. (2021). Optimization of autonomous agent routes in logistics warehouse. *International Journal of Electronics and Telecommunications, 67*, 559–564.
47. Sokolov, I., & Turkin, I. (2018). Resource efficient data warehouse optimization. In *2018 IEEE 9th international conference on dependable systems, services and technologies (DESSERT)* (pp. 491–495).
48. Raghuram, P., & Singh, A. (2020). Warehouse optimization using demand data analytics. *International Journal of Business Information Systems, 1*, 1.
49. Srinivas, M., & Patnaik, L. M. (1994). Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics, 24*, 656–667.
50. Hassanat, A., Almohammadi, K., Alkafaween, E., Abunawas, E., Hammouri, A. M., & Prasath, V. B. S. (2019). Choosing mutation and crossover ratios for genetic algorithms–a review with a new dynamic approach. *Information, 10*, 390.
51. Indyk, P., & Motwani, R. (1998). Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Symposium on the theory of computing*.
52. Shehory, O., Sycara, K. P., & Jha, S. (1997). Multi-agent coordination through coalition formation. In *ATAL*.
53. Papoudakis, G., Christianos, F., Schäfer, L., & Albrecht, S. V. (2020). Comparative evaluation of multi-agent deep reinforcement learning algorithms. CoRR, arXiv:abs/2006.07869
54. Myerson, J., & Green, L. (1995). Discounting of delayed rewards: Models of individual choice. *Journal of the experimental analysis of behavior, 64*, 263–76, 12.
55. Budish, E. (2011). The combinatorial assignment problem: Approximate competitive equilibrium from equal incomes. *Journal of Political Economy, 119*, 1061–1103.
56. Bezerra, L. C. T., López-Ibáñez, M., & Stützle, T. (2018). A large-scale experimental evaluation of high-performing multi- and many-objective evolutionary algorithms. *Evolutionary Computation, 26*(4), 621–656, 12.
57. Devlin, S., Yliniemi, L., Kudenko, D., & Tumer, K. (2014). Potential-based difference rewards for multiagent reinforcement learning. In *Proceedings of the 2014 international conference on autonomous agents and multi-agent systems, AAMAS '14* (pp. 165–172). Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.
58. Buşoniu, L., Babuska, R., & De Schutter, B. (2008). A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 38*, 156–172.
59. Bidram, A., Lewis, F. L., & Davoudi, A. (2014). Distributed control systems for small-scale power networks: Using multiagent cooperative control theory. *IEEE Control Systems, 34*, 56–77.
60. Ren, W., & Beard, R. W. (2005). Consensus seeking in multiagent systems under dynamically changing interaction topologies. *IEEE Transactions on Automatic Control, 50*, 655–661.
61. Paredes, A., & del Olmo Martínez, R. The social dimension of economics and multiagent systems.
62. Todorov, E., Erez, T., & Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE international conference on intelligent robots and systems* (pp. 5026–5033). IEEE.
63. Freeman, C. D., Frey, E., Raichuk, A., Girgin, S., Mordatch, I., & Bachem, O. (2021). Brax–a differentiable physics engine for large scale rigid body simulation.
64. Kutschinski, E., Uthmann, T., & Polani, D. (2003). Learning competitive pricing strategies by multiagent reinforcement learning. *Journal of Economic Dynamics and Control, 27*(11), 2207–2218. Computing in economics and finance.

Authors and Affiliations

Andreas Kallinteris[1] · Stavros Orfanoudakis[1,2] · Georgios Chalkiadakis[1]

✉ Andreas Kallinteris
akallinteris@tuc.gr

Stavros Orfanoudakis
s.orfanoudakis@tudelft.nl

Georgios Chalkiadakis
gehalk@intelligence.tuc.gr

[1] School of ECE, Technical University of Crete, 73100 Chania, Greece

[2] Intelligent Electrical Power Grids (IEPG) Group, Delft University of Technology, 2628 CB Delft, The Netherlands