Distributed Online Rollout for Multivehicle Routing in Unmapped Environments

Jamison W Weber
Arizona State University
Tempe, Arizona, United States
jwweber@asu.edu

Dhanush R Giriyan
Arizona State University
Tempe, Arizona, United States
dgiriyan@asu.edu

Devendra R Parkar
Arizona State University
Tempe, Arizona, United States
dparkar1@asu.edu

Dimitri P Bertsekas
Arizona State University
Tempe, Arizona, United States
dbertsek@asu.edu

Andréa W Richa
Arizona State University
Tempe, Arizona, United States
aricha@asu.edu

ABSTRACT

We consider a generalization of the well-known multivehicle routing problem: given a network, a set of agents occupying a subset of its nodes, and a set of tasks, we seek a minimum cost sequence of movements subject to the constraint that each task is visited by some agent at least once. The classical version of this problem assumes a central computational server that observes the entire state of the system perfectly and directs individual agents according to a centralized control scheme. In contrast, we assume that there is *no centralized server* and that each agent is an individual processor with *no a priori knowledge of the underlying network, including the locations of tasks and other agents.* Moreover, our agents possess *strictly local communication and sensing capabilities* (restricted to a fixed radius), aligning more closely with several real-world multiagent applications. We present a *fully distributed, online, and scalable reinforcement learning algorithm* for this problem whereby agents self-organize into local clusters to which they independently apply a multiagent rollout scheme. We demonstrate empirically via extensive simulations that there exists a critical sensing radius beyond which the distributed rollout algorithm begins to improve over a greedy base policy. This critical sensing radius grows proportionally to the $\log^*$ function of the size of the network and is therefore a small constant in practice. Our decentralized reinforcement learning algorithm achieves approximately a factor of two cost improvement over the base policy for a range of radii between two and three times the critical sensing radius. In addition, we observe in simulations that our distributed algorithm requires *exponentially fewer computational resources* than the centralized approach, at only a *small constant factor detriment in cost.* We validate our algorithm through physical robot experiments in continuous space and show that the resulting behavior reflects the discrete simulations.

KEYWORDS

Reinforcement Learning; Distributed Computing; Multivehicle Routing

Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – 10, 2024, Auckland, New Zealand. © 2024 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

ACM Reference Format:
Jamison W Weber, Dhanush R Giriyan, Devendra R Parkar, Dimitri P Bertsekas, and Andréa W Richa. 2024. Distributed Online Rollout for Multivehicle Routing in Unmapped Environments. In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024*, IFAAMAS, 9 pages.

1 INTRODUCTION

In this work, we consider a generalization of the well-studied multivehicle routing problem (MVRP). The MVRP involves a set of agents (vehicles) and a set of tasks each occupying a position on an underlying network. The objective is to compute a sequence of movement decisions for each vehicle such that the total number of vehicle movements is minimized, subject to the constraint that each task is visited at least once by some agent. Classically, these sequences are computed by a centralized controller with complete state information that may freely direct all agents according to the control sequences it computes.

Our generalization addresses scenarios where the *network topology, including the initial positions of agents and task locations, is not known a priori*, and where a *centralized controller is not available*, as is potentially the case in many real-world scenarios, such as minefield disarmament [16], post-disaster search and rescue [21], and unmanned aerial vehicle navigation [5]. In such settings, each agent must be in turn capable of performing individual computations and of *local sensing and inter-agent communication*, both limited to a radius around the agent's location in the network (e.g. as in packet radio networks). Since no centralized computation is available, local communication between agents is necessary for coordination and information exchange. Since sensing is limited, exploration is required to locate the tasks before completing them. We refer to this as the *Unmapped Multivehicle Routing Problem with Local Constraints (UMVRP-L).* Note that MVRP can be viewed as a special case of UMVRP-L where the sensing radius is infinite.

The UMVRP-L is challenging for agent coordination algorithm design. Algorithmically, one may consider a reinforcement learning solution, as has been done with other MVRP variants [33], but as the underlying network is unknown, methods that require offline computation such as traditional Q-learning [35], policy approximation (via the training of parametric architectures) [12], or executing policy iteration to near convergence [34] may not be applicable. Moreover, even online Q-learning requires extensive exploration

over a single trajectory to learn Q-values, and also frequently revisits states to ensure convergence [30]. This exploration factors into the total cost of solving the problem.

This realization suggests rollout [10] as a natural candidate, as it is a reinforcement learning algorithm that (under certain conditions) admits a fully online solution without requiring precomputation or iterative convergence, and produces reliably favorable results due to its fundamental connections to Newton's method [9–11]. However, rollout is a centralized control scheme that typically assumes information is universally shared among agents and a central dispatch (see *classical information pattern* [10]), and hence we present a decentralized version of rollout to address UMVRP-L. In general, the rollout algorithm requires a base policy, which provides a benchmark for comparison. For our distributed modification of rollout specifically, this comparison allows us to evaluate the impact of *communication* and *coordination* in our distributed framework: Communication refers to agents sharing sensed information, whereas coordination refers to agents organizing in such a way that they collectively solve a problem intelligently.

1.1 Our Contributions

We present a distributed (decentralized) version of the Multiagent Rollout algorithm from [10], which we call Decentralized Multiagent Rollout (DMAR). In this approach, we assume each agent runs DMAR independently and in parallel with other agents. DMAR uses simple local rules to induce iterative self-organization of the agent set into temporary clusters of bounded size. These clusters generate control sequences locally using multiagent rollout (equipped with a greedy nearest-neighbor base heuristic [10]) and execute them before systematically dissolving. Agents alternate between self-organizing into clusters and randomly searching the environment for tasks not visible to any cluster, making DMAR applicable to *fully unmapped environments*, unlike the general multiagent optimization literature.

We formally prove that DMAR is *probabilistically complete* [7], i.e. for any solvable instance, the probability that *DMAR correctly solves UMVRP-L* converges to unity as the running time tends to infinity. We also show that each *round* of DMAR terminates in constant time and that the *expected number of rounds required to solve the instance is bounded by* $O(N^2)$, where N is the number of nodes in the network. Moreover, we conduct extensive simulations that compare the costs incurred by DMAR with those of a modification of DMAR that we call *the greedy-exploration policy*—a base policy that strictly uses a greedy nearest-neighbor base heuristic (without application of multiagent rollout) within the clusters generated. Note that the greedy-exploration policy does not involve coordination of the control policies within each cluster. In these experiments, we modulate the sensing radius over a wide variety of network topologies and sizes, the number of tasks, and agent-to-task ratios.

We show that there is a *critical sensing radius* beyond which DMAR begins to improve over the greedy-exploration policy: The critical sensing radius is closely approximated by $\log_2^*(N)$,[1] and is therefore a small constant in practice regardless of network size. We show empirically that for sensing radii between $2\log_2^*(N)$ and $3\log_2^*(N)$ (we call those the *effective radii*), DMAR generates trajectories of roughly a *half the cost of those produced by the base policy* on average. We compare the costs of DMAR to that of the centralized multiagent rollout algotithm [10]. For the effective radii range, DMAR shows an *exponential improvement in computational costs over the centralized multiagent rollout algorithm*, while incurring just *a constant factor (approximately 3) increase in movement costs*.

Lastly, we validate our algorithm through physical robot experiments in continuous space using the Robotarium platform [39] as a proof-of-concept. The results reflect what we observed in discrete-space simulations, demonstrating the algorithm's robustness to sensor noise, which would underscore the potential for actual deployments of the algorithm in scenarios where it is not reasonable to assume that a complete mapping of tasks and agents exists.

1.2 MVRP and Other Related Work

A discussion on reinforcement learning techniques already appeared in the introduction, so here we focus more on the MVRP literature and agent-based distributed systems. As the MVRP is NP-hard [36] (even when the network is planar [20]), several heuristics have been proposed (e.g., [10, 28]). Researchers have considered many fully observable and centralized variants that introduce further constraints and complexities such as limited vehicle capacity, heterogeneous capacities, route length limits, time windows when tasks may be completed, multiple depots, multi-criteria objective functions, stochastic time and transition variants, and demand constrained and weighted MVRP, e.g. [14, 23, 24].

Most relevant to UMVRP-L is probably the *dynamic MVRP* variant [24, 31], where certain input information such as the network, vehicle availability, or task locations may not be known in advance, but becomes available over time (perhaps probabilistically). Dynamic MVRP assumes a single centralized dispatcher with instantaneous access to information as it becomes available, instant communication with all vehicles, and persistent access to all previously available information—all of which we do not assume in this work. Although UMVRP-L has commonalities with dynamic MVRP, it is fundamentally different due to its decentralized nature and strict local constraints. To the best of our knowledge, variants of vehicle routing where a centralized planner is not available have not yet been considered by researchers, and hence, no benchmark algorithms for UMVRP-L exist in the literature.

AI researchers have also considered multiagent problems where the full state is not directly observable to the agents, but the proposed solutions in this domain require centralized computation—often executed offline (e.g. [6, 13, 19, 22, 25]). A formulation known as a *decentralized partially observable Markov decision process* (Dec-POMDP) was introduced in [8] that combines notions from distributed computing and optimal control in partially observable environments, but optimally solving Dec-POMDPs was shown to be intractable. In these formulations, the environment is partially observable to each agent, but the information sensed by each agent is not universally shared. Approximate solutions to Dec-POMDPs were explored in [1–3], but ultimately relied on intensive offline and centralized simulations.

More broadly, there have been several recent advances in the field of distributed computing on engineering collective behavior

[1] $\log_2^*(N) = 1 + \log_2^*(\log_2(N))$, for $N > 1$ and $\log_2^*(N) = 0$, for $N \leq 1$. Note that, e.g, $\log_2^*(N) \leq 5$ for any $N \leq 10^{10000}$.

for systems of agents. Such agents are typically restricted to only local computations and interactions with other agents to achieve a desired system-wide behavior, as in population protocols [4] and programmable matter (e.g. [15, 17, 37, 40, 41]).

2 MODEL AND PRELIMINARIES

In general, one assumes a regular discretization of 2D space for the MVRP, such as a 2-dimensional *rectangular grid graph* (as in [34]), as we do in this work. An instance of the MVRP consists of a rectangular grid graph $G(V, E)$ (e.g. in Figure 1) on N nodes, a set S of m *agents*, where each agent occupies a node in the graph (co-location is permitted), and a subset τ of nodes that represent *task* locations in G. We also assume there is a subset O of nodes that agents may not occupy, which allows us to represent *obstacles* in the environment (represented by the black squares in Figure 1). Moreover, each agent has a unique ID. This model operates in discrete time steps, during which an agent may move to a node adjacent to its current location. Whenever an agent visits a task node, that task is considered *completed*, and is removed from the network. At each time step, a unit cost is incurred for each agent that moves from one node to an adjacent node. A control sequence Y_i for agent i is given by the sequence of nodes $v_{(i,1)}, v_{(i,2)}, \ldots$ visited by the agent. A solution Y is given by m sequences $Y_1, \ldots, Y_m$ of controls corresponding to each agent in S. The cost $C(Y)$ of a solution Y is the sum of the m control sequence lengths, i.e. $C(Y) = \sum_{Y_i \in Y} |Y_i|$. A solution Y is optimal whenever $C(Y)$ is minimum.

The UMVRP-L is a generalization of MVRP where G, O, S, τ, are not known a priori and no centralized computation is permitted. We assume that each agent has some on-board computational capability and enforce that any computation must be executed by some agent, as opposed to a central controller. At any given time, each agent a occupies a position $\rho(a) \in V$. We also define the *view* of an agent a, denoted as $v(a)$, to be the subgraph of the grid induced by $\rho(a)$ and *all* nodes (including obstacles) within at most k hops of $\rho(a)$, where k is an integral radius parameter. An agent's view describes what it is able to sense from its location (i.e. obstacles, neighboring nodes, agents, and tasks). Depending on the application context, there may be other (more appropriate) choices for the definition of a view, such as those where obstacles restrict what an agent can observe. We provide a discussion of the impact of various view definitions in Appendix-A [38]. Let $v(S') = \bigcup_{a \in S'} v(a)$, and refer to this as the *view* of an agent set $S' \subseteq S$.

Our distributed model operates under *parallel synchronous time steps*. That is, at each discrete time step t, all agents are *activated* simultaneously in parallel, and thus may only act upon information communicated at time $t - 1$. An *agent activation* may consist of either moving, performing some computation, or simply waiting. For an agent a, the input of a computation performed by a is limited to $v(a)$ (including the memories of the agents therein), and any agent $b \in v(a)$ (including a) may have their memory modified directly by a as an output to that computation.

Rollout and Multiagent Rollout. From a centralized perspective, one can view an instance of MVRP as an infinite horizon dynamic programming problem that can be solved approximately in an online setting using *approximation in value space*, of which the rollout algorithm is a special case [10]. That is, we approximate the *cost-to-go* terms of the Bellman equations via online heuristics. Denote Ω as the state space representing all possible agent and task configurations over G, plus an absorbing terminal state representing the completion of all tasks. Let $U(x)$ denote the set of m-dimensional control vectors permissible at state $x \in \Omega$. A system dynamics function $f(x, u)$ returns the resulting state obtained from application of control $u \in U(x)$ at state $x \in \Omega$. Lastly, denote $g(x, u)$ as the cost function indicating the total number of vehicle movements at state x given control u. Rollout generates an approximately optimal state and control sequence $x_0, \tilde{u}_0, x_1, \tilde{u}_1, \ldots$ where each successive control $\tilde{u}_s$ at state x_s is determined by

$$\tilde{u}_s \in \arg \min_{u \in U(x_s)} \{g(x_s, u) + \tilde{H}(f(x_s, u))\} \tag{1}$$

and results in arrival at state x_{s+1}. Here $\tilde{H}(f(x_s, u))$ refers to the approximate cost-to-go obtained from simulation of a heuristic (also known as a *base policy*).

Note that the minimization operation in Equation 1 is exponential in m, however, a recent reformulation of a multiagent problem to an equivalent problem with a larger state space, but simpler control space was described by Bertsekas in [10]. This reformulation was the basis for substantial simplification of rollout and policy iteration algorithms for multiagent problems. The resulting algorithm (called *multiagent rollout*) dramatically reduces the computational complexity of each minimization operation of the standard rollout algorithm (from exponential to linear in the number of agents), while maintaining its cost improvement properties. This exponential gain arises from a sequential rollout control computation on an agent-by-agent basis, where the control for each agent is computed based on a partial list of controls computed for each agent earlier in the sequence, hence greatly restricting the number of multi-component controls considered. Specifically, the control u^a for each agent a is computed in a sequence $u^{a_1}, u^{a_2}, \ldots, u^{a_m}$ such that when u^{a_i} is computed, each u^{a_j} for $1 \leq j < i$ has been determined, and for any u^{a_j} where $i < j \leq m$, and for all future controls, a base policy is used to estimate the cost-to-go in Equation 1. See [10] for a comprehensive description.

3 DECENTRALIZED MULTIAGENT ROLLOUT

In this section, we present and analyze our Decentralized Multiagent Rollout Algorithm (DMAR), described as a series of phases that, when executed in sequence and repeated, will solve the UMVRP-L. DMAR is a fully distributed and local algorithm that applies multiagent rollout from [10] locally, and therefore can handle unmapped environments. In addition to their (unique) IDs, DMAR assumes that the agents know a constant parameter ψ (common to all agents), which will determine an upper bound on the size of the clusters formed by DMAR. During its execution, the algorithm will build trees of agents, each of which corresponds to a current cluster, and so each agent also maintains a parent pointer, a constant number c of child pointers, and various flags with values T,F used for message passing. At a high level, a *round of DMAR* consists of the following sequence of *phases*, which the agents collectively repeat:

(1) **Self-Organizing Agent Clusters (SOAC)**: Agents self-organize into clusters of constant size $c^{O(\psi)}$. For each cluster K, the agents in K will locally form a tree $\mathcal{T}_K$, whose root

will become the cluster leader. If two agents share an edge in $\mathcal{T}_K$, then they appear in their respective views.

(2) **Local Map Aggregation (LMA)**: For each cluster K, agents in K route their views through $\mathcal{T}_K$ to the cluster leader.

(3) **Team-Restricted Multiagent Rollout (TMAR)/Execute Movement (EM)**: For each cluster K, the leader computes a set $\mathcal{R}$ of control sequences (one for each agent in K) using multiagent rollout on the assembled view information. $\mathcal{R}$ is then broadcast over $\mathcal{T}_K$. If an agent does not belong to a cluster, it follows a random trajectory of bounded length. Else, each agent moves along its assigned control trajectory, after which it unassigns itself from its cluster.

We will show that each *round* of DMAR terminates in constant time, and that the number of rounds required for DMAR to solve an instance of UMVRP-L is polynomially bounded in expectation (Theorem 3.4). A pseudocode description of a round of DMAR is given in Algorithm 1. We denote $\mathcal{L}(\psi) = \frac{3}{2}(\psi - 2)$ as the maximum possible height of any agent tree (see Lemma 3.1) and reference the function $\mathrm{MR}(\mathcal{M})$, which takes an instance $\mathcal{M}$ of MVRP and returns a set of control sequences $\mathcal{R}$ as computed by multiagent rollout (see Section 2). Note that each agent runs DMAR individually on their respective processor, but all agents remain synchronized in their execution of DMAR. We achieve this by ensuring that each phase is executed using exactly the same number (a function of ψ) of operations regardless of which agent is performing the computation. For example, an agent that is not in a cluster will wait approximately $2 \cdot \mathcal{L}(\psi)$ time steps for Lines 29-42 before continuing.

Self-Organizing Agent Clusters *(Algorithm 1, Lines 2-28).* To achieve the clustering, agents that see tasks become temporary cluster *leaders* via a local leader election scheme. The clusters then grow by iteratively appending nearby agents (limited by parameter ψ). At the start of SOAC, each agent sets a leader flag (*ldrFlag*) to TRUE (T) if it sees a task. Then, if an agent a sees a task while simultaneously seeing other agents that each see tasks, a disqualifies itself from becoming a leader if any such agent has a larger ID value than a's by setting its leader flag to FALSE (F). Any remaining agents after the above process form leaders of new unique clusters. Next, SOAC iteratively appends agents to existing clusters over multiple time steps. At iteration i of the loop beginning in Line 6, if there exists an agent a that does not belong to a cluster, a searches its view for any agent a' that already belongs to some cluster K' and joins K' by becoming a child of a'. Consequently, as each cluster K grows, a bi-directional (through parent and child pointers) tree structure $\mathcal{T}_K$ is maintained that manages the flow of messages in K. Moreover, when a joins a cluster K, it obtains K's cluster ID from its parent in $\mathcal{T}_K$, which corresponds to the ID of the leader of K.

We found in preliminary simulations that communication between adjacent clusters generally leads to significantly better solutions. As such, for adjacent clusters, SOAC employs a mechanism to combine them into potentially larger clusters called *cluster-join* (Lines 13-27). For any agent a that is not in a cluster at iteration i after the appending process (Lines 9-11), if a sees other agents in distinct clusters, a becomes the leader of a new (possibly larger) cluster that attempts to include all agents in neighboring clusters. Agent a sends messages (*joinMessage*) to all such agents in its view, which are then broadcast throughout the respective agent trees.

Algorithm 1 Local DMAR round protocol for agent a

1: Set $a.msgFlag \leftarrow a.ldrFlag \leftarrow$ F; Set $a.clusterID \leftarrow a.joinMessage \leftarrow a.children \leftarrow a.parent \leftarrow$ NULL.
2: **if** there is a task in $v(a)$, set $a.ldrFlag \leftarrow$ T. ▷ begin SOAC
3: **if** $a.ldrFlag$ and $\exists x \in v(a) : x.ldrFlag$ and $x.ID > a.ID$ **then**
4: set $a.ldrFlag \leftarrow$ F.
5: **if** $a.ldrFlag$ **then** set $a.clusterID \leftarrow a.ID$.
6: **repeat** ▷ form initial clusters
7: **if** $a.clusterID =$ NULL **then**
8: **if** there is an agent $x \in v(a)$ with $x.clusterID \neq$ NULL
9: and $|x.children| < c$ **then**
10: Append a to $x.children$; set $a.parent \leftarrow x$.
11: Set $a.clusterID \leftarrow x.clusterID$.
12: **else**
13: **if** there are agents $x_1, x_2, \ldots, x_r \in v(a)$ from distinct clusters, where $r \leq c$ **then** ▷ cluster-join
14: Set $a.ClusterID \leftarrow a.ID$.
15: **for** each x_i **do**
16: **try** set $x_i.joinMessage \leftarrow a.ID$.
17: ▷ Note that x_i may reject the join request.
18: **repeat**
19: **if** $a.joinMessage = x.ID$ **then**
20: Set $a.ClusterID \leftarrow x.ClusterID$.
21: **for** each child b of a **do**
22: **if** $x \neq b$, **try** set $b.joinMessage \leftarrow a.ID$.
23: **if** $a.parent \neq x$ **then**
24: **try** set $a.parent.joinMessage \leftarrow a.ID$.
25: Set $a.parent \leftarrow x$; append a to $x.children$.
26: Set $a.children \leftarrow a.joinMessage \leftarrow$ NULL.
27: **until** $\mathcal{L}(\psi)$ (synchronized) iterations
28: **until** $\lceil \log_2 \psi \rceil$ (synchronized) iterations
29: **if** $a.clusterID \neq$ NULL **then** ▷ begin LMA
30: Construct map $\mathcal{M} = v(a)$.
31: **If** $a.parent.clusterID = a.parent.ID$, set $a.msgFlag \leftarrow$ T.
32: **repeat**
33: **if** $a.msgFlag$ **then**
34: Set $\mathcal{M}(a) \leftarrow \mathcal{M}(a) \cup \mathcal{M}(a.parent)$, preserving coordinates relative to $\mathcal{M}(a.parent)$.
35: **if** $a.children \neq$ NULL, set $a.msgFlag \leftarrow$ F.
36: **for each** child b, set $b.msgFlag \leftarrow$ T.
37: **until** $\mathcal{L}(\psi)$ (synchronized) iterations
38: **repeat**
39: **if** $a.msgFlag$ and $a.clusterID \neq a.ID$ **then**
40: Set $\mathcal{M}(a.parent) \leftarrow \mathcal{M}(a) \cup \mathcal{M}(a.parent)$.
41: Set $a.parent.msgFlag \leftarrow$ T; $a.msgFlag \leftarrow$ F.
42: **until** $\mathcal{L}(\psi)$ (synchronized) iterations
43: **if** a is the leader of a cluster K **then** ▷ begin TMAR/EM
44: Compute $\mathcal{R} \leftarrow \mathrm{MR}(\mathcal{M}(K))$ and send $\mathcal{R}$ to each child.
45: **repeat**
46: **if** $a.clusterID \neq$ NULL and a has received $\mathcal{R}$ from parent, **then** send $\mathcal{R}$ to each child of a.
47: **until** $\mathcal{L}(\psi)$ (synchronized) iterations
48: **if** $a.clusterID \neq$ NULL **then** move a according to $\mathcal{R}^{a.ID}$.
49: **else** a executes at most $\lambda(\psi)$ controls, each selected uniformly at random until there is a task in $v(a)$.

These messages request that its recipients adopt the new cluster ID (that of agent a) and reconfigure their tree pointer structure such that their tree is rooted at a. If an agent receives multiple cluster reassignment messages simultaneously, it will disregard all but one (chosen arbitrarily, see line 16). Note that because of the possibility that an agent may receive multiple cluster reassignment requests simultaneously, not all agents of these trees may join the new cluster rooted at a, since some may join other clusters in this manner. However, in simulations we observed this generally resulted in larger clusters, which led to greatly improved performance. Each agent continues propagating join messages for $\mathcal{L}(\psi)$ time steps per iteration. After a SOAC run terminates, the process yields a collection of clusters (and possibly some agents that belong to no clusters). Generally, if the would-be parent of an agent a attempting to join a cluster has no available child pointers, then a simply does not join the cluster. Figure 1 provides a visualization of a SOAC execution. Lemma 3.1 guarantees properties of the tree structure of each cluster formed by SOAC, and bounds the running time.

Lemma 3.1. *Let $K \subseteq S$ be an arbitrary cluster of agents in G obtained at the end of a round of SOAC. Then (i) K forms an agent tree $\mathcal{T}_K$ rooted at a leader ℓ with height at most $\mathcal{L}(\psi) = O(\psi)$ and $|\mathcal{T}_K| = c^{O(\mathcal{L}(\psi))}$ agents; (ii) all agents in $\mathcal{T}_K$ associate their membership in K with ℓ; and (iii) SOAC terminates in $O(\psi \log_2 \psi)$ time steps.*

Proof. (i) Each tree is formed around a single leader. An unassigned agent appends itself as a child of a single parent node already in $\mathcal{T}_K$, for some current cluster K, adopting the cluster ID K of its parent (which can be traced back to the leader of K). Thus $\mathcal{T}_K$ will form a single acyclic connected component, for each K. In the cluster-join process (initiated by an agent b), the parent pointer of an agent a will always point to the agent from which it received (and accepted) a join request, which can likewise be traced back to b. As agent a removes its child pointers after propagating the request, it becomes a leaf of the new tree, and any of its former children may undergo the same process in the next loop iteration. At a loop iteration i, we claim that the height of the largest cluster tree first increases by at most one (Lines 9-11) and then at most doubles (plus one for the joining agent b) during the cluster-join process. This follows since for any agent x in $\mathcal{T}_K$, the neighborhood of x in $\mathcal{T}_K$ (i.e. not including b) after the cluster-join process is an induced subgraph of x's previous neighborhood, and any neighbors it loses are no longer in $\mathcal{T}_K$. As x does not gain new neighbors in $\mathcal{T}_K$, the longest path in $\mathcal{T}_K$ is either the same or some shorter path (in the case where not all of K remains in the tree) after pointer reassignment. The longest possible path in any undirected rooted tree $\mathcal{T}$ is at most twice the height of $\mathcal{T}$. Thus, the maximum tree height at iteration i is given by $\mathcal{L}(i) = 0$ if $i = 1$, and $\mathcal{L}(i) \leq 2(\mathcal{L}(i-1)+1)+1$ for $i > 1$. The solution of this relation is $\frac{3}{2}(2^i - 2) = O(2^{\log_2 \psi}) = O(\psi)$. The maximum number of agents in $\mathcal{T}_K$ is $\sum_{j=1}^{\mathcal{L}(\psi)} c^j = \frac{c^{\mathcal{L}(\psi)+1}-1}{c-1} \leq c^{O(\mathcal{L}(\psi))}$. The height bound on an agent tree implies waiting $\mathcal{L}(\psi)$ time steps per iteration is enough time for join messages to propagate through the tallest tree, and so all agents in $\mathcal{T}_K$ will agree on the leader ID, implying (ii). Counting shows that SOAC terminates in $O(\psi \log \psi)$ time steps. □

Local Map Aggregation *(Algorithm 1, Lines 29-42).* At the end of SOAC, a subset of the agents is organized into clusters, for each of which there exists a respective leader. In the second phase, the objective is that for each independent cluster K with leader ℓ, agent ℓ obtains a map of K, which reflects the collective view of the cluster ($v(K)$). Moreover, this map has a relative coordinate labelling scheme whereby the coordinates of ℓ are $(0, 0)$. At a high level, agents in K convey their views to their neighbors in $\mathcal{T}_K$. These views then aggregate until ℓ obtains a view of the entire cluster.

An agent a possesses a *map* structure $\mathcal{M}(a)$ that includes $v(a)$, as well as possibly the views of other agents (initially $\mathcal{M}(a) = v(a)$). Note that their may be agents in the view of a that do not belong to the cluster of which a is a member, but a disregards these. A *complete map* $\mathcal{M}(K)$ of a cluster K includes the union of views of every agent in K, i.e. $\mathcal{M}(K) = v(K)$. For two maps $\mathcal{M}$ and $\mathcal{M}'$, we use $\mathcal{M} \cup \mathcal{M}'$ to denote the component-wise union of the maps. Agent ℓ will

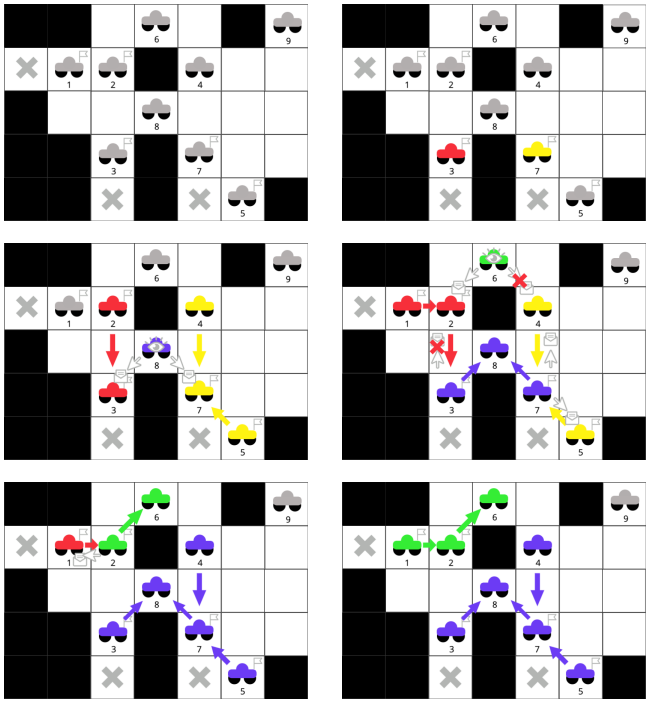

Figure 1: SOAC Execution. Read left to right, top to bottom: Cars represent agents. Gray crosses are tasks. Agent colors indicate cluster membership; gray agents are in no cluster. IDs are indicated below each agent. Flags indicate which agents see tasks. Assume $k = 2$, $\psi = 4$. (top left) No agent is in a cluster; 1,2,3,5,7 see tasks. (top right) 1,2,5 see agents with larger IDs who see tasks, so disqualify themselves. 3,7 become leaders of new clusters. (middle left) Iteration 1 begins. 2,4,5 join clusters, colored arrows indicate tree pointers. 8 sees agents in different clusters and initiates a join with messages. (middle right) 3,7 join the purple cluster, send messages to children to request reassignment. 2 rejects simultaneous request from 3. 6 initiates cluster join, 4 rejects request. (bottom left) Iteration 2 begins. 2 joins the green cluster. 4 favors the purple cluster, 5 joins the purple cluster. 2 sends request to 1. (bottom right) 1 joins green cluster. 9 was too distant.

initiate the propagation of messages down to the leaves of $\mathcal{T}_K$. Agent ℓ generates a map $\mathcal{M}(\ell) = \nu(\ell)$, which has nodes labelled with relative coordinates such that $\rho(\ell) = (0, 0)$. Agent ℓ passes this map to each child a, and agent a sets $\mathcal{M}(a) \leftarrow \mathcal{M}(a) \cup \mathcal{M}(\ell)$. During this union operation, agent a extends the relative coordinates of $\mathcal{M}(\ell)$ to $\mathcal{M}(a) \cup \mathcal{M}(\ell)$. This message passing process from parent to child repeats using a message flag (*msgFlag*). After a sufficient amount of time passes, each leaf a sends $\mathcal{M}(a)$ (which represents a union of maps obtained from the path from ℓ to a in $\mathcal{T}_K$) to its respective parent b, and updates its map $\mathcal{M}(b) \leftarrow \mathcal{M}(b) \cup \mathcal{M}(a)$ (again preserving relative coordinates). This process then iterates $\mathcal{L}(\psi)$ times from child to parent where then ℓ receives a complete map. Agents repeatedly propagate messages long enough for the height of the tallest tree to be traversed twice (down then up) to preserve a relative coordinate scheme that is common to all agents in K. If $\mathcal{M}(K)$ contains tasks that are unreachable by some agent, ℓ disregards their respective components, and if this results in a map without tasks, the cluster is dissolved via broadcast of messages that cause agents to reset their cluster associations and pointers. We omit this description from Algorithm 1 for brevity. The lemma below guarantees correctness and running time.

Lemma 3.2. *Given $\mathcal{T}_K$, local map aggregation provides the leader ℓ of the cluster K with a complete map $\mathcal{M}(K) = \nu(K)$ with all nodes labelled relative to $\rho(\ell) = (0, 0)$ in $O(\psi)$ time steps.*

Proof. As the size of any cluster is constant, and an agent a in cluster K will disregard any agent not in K, $\mathcal{M}(a)$ has constant size and is generated in constant time. By Lemma 3.1, the maximum height of $\mathcal{T}_K$ is $\mathcal{L}(\psi)$, and so $O(\mathcal{L}(\psi))$ is a sufficient time so that the leaves of $\mathcal{T}_K$ receive partial cluster maps from ℓ. Likewise, the broadcast of map information from the farthest leaf to the root requires $O(\mathcal{L}(\psi))$ time steps. As each message pass from an agent a to an agent b produces a union of $\mathcal{M}(a)$ and $\mathcal{M}(b)$, after $O(\mathcal{L}(\psi)) = O(\psi)$ time steps, ℓ possesses a complete map of K. Note that the number of messages passed from a parent to its children is constant since c is constant. By the labelling scheme given in LMA, the complete map obtained by leader ℓ is labelled with coordinates relative to $\rho(\ell) = (0, 0)$. This follows since the map originates from ℓ, where it is labelled with relative coordinates to $\rho(\ell) = (0, 0)$, and every map union operation appends nodes to a map that contains the original map of ℓ, so the relative labelling scheme is preserved in the complete map. □

***Team-Restricted Multiagent Rollout and Execute Movement** (Algorithm 1, Lines 43-49).* TMAR describes how to apply multiagent rollout from [10] to an independent cluster K while respecting all of the local sensing and communication restrictions of our model. After the sequential execution of SOAC followed by LMA, each cluster K is structured as a spanning tree $\mathcal{T}_K$ rooted at the leader agent ℓ, who possesses a complete map $\mathcal{M}(K)$. Agent ℓ executes multiagent rollout on $\mathcal{M}(K)$ with a greedy nearest-neighbor base policy (i.e. each agent moves toward its respective nearest task) to obtain control sequences for each agent in K, which terminates in constant time since the size of $\mathcal{M}(K)$ is bounded by a constant due to the constant height and branching factor of any agent tree. Agent ℓ then initiates a broadcast of these sequences (via parent-to-child message passing in $\mathcal{T}_K$), each of which is associated with a specific agent ID. After enough iterations have passed for each agent a to receive its corresponding control sequence, a will simply execute its control sequence via movement (EM). To maintain synchronization between all clusters, a will wait an additional number of time steps until it reaches a maximum control sequence length $\lambda(\psi) = O([\mathcal{L}(\psi)]^4)$—a function on ψ that returns a bound on the maximum route length for a greedy nearest-neighbor heuristic applied to $\mathcal{M}(K)$. For any agent a not in a cluster, instead of following a computed control sequence, a executes a (non-empty) sequence of controls generated uniformly at random of length at most $\lambda(\psi)$ and waits as soon as it sees a task, thereby exploring the network.

The theorem below shows that, in constant time, a round of DMAR separates a subset of the agents into clusters, for each of which control sequences are generated from multiagent rollout locally and executed. Agents that are not members of clusters perform random walks of at most constant length in each run of EM.

Theorem 3.3. *During a round r of DMAR, the agent set is partitioned into members of clusters and non-members. Within $O(\psi)$ time steps, each agent x that is in a cluster K executes a control sequence $\mathcal{R}^x$ that is generated from multiagent rollout with a greedy base policy applied to $\mathcal{M}(K)$, and executes $\mathcal{R}^x$ within $O(\psi^4)$ time steps. If x is not in a cluster, x performs a random walk of length at most $\lambda(\psi) = O(\psi^4)$. Lastly, round r terminates in constant time.*

Proof. As $O(\psi)$ additional time is required for message broadcast in the event K is dissolved at the end of LMA (due to the tree height), by Lemmas 3.1 and 3.2, the agent set is partitioned into agents that are members of clusters and those who are not in constant time. Those that are in a cluster K are associated with an agent tree $\mathcal{T}_K$, the root of which possesses a map $\mathcal{M}(K)$ of K. By Lemma 3.1(i) and since k is constant, the radius of $\mathcal{M}(K)$ is bounded by $O(\psi)$, and hence the area of $\mathcal{M}(K)$ is bounded by $O(\psi^2) = O(1)$. The number of agents in K is bounded by a constant $(c^{O(\mathcal{L}(\psi))})$. As the area of $\mathcal{M}(K)$ is of constant size, it can contain at most a constant number of tasks (i.e. at most one per node). Hence, any multiagent rollout computation on a map $\mathcal{M}(K)$ will terminate in constant time. Consequently, by Lemma 3.1(i) and according to TMAR, each agent in a cluster K obtains a control sequence computed from multiagent rollout applied to $\mathcal{M}(K)$ in a constant number of time steps and moves accordingly.

In the worst case, there may be a task at every node, and a route may visit every task with a cost incurred proportional to the diameter of the cluster. This implies the longest route for a cluster K is bounded by $\lambda(\psi) = O(\psi^4) = O(1)$ since the diameter of the cluster is bounded by its area. Since the base policy is greedy, multiagent rollout can perform no worse than it (see cost-improvement properties [10]), and hence can never produce a trajectory longer than $\lambda(\psi)$. As each agent spends a constant number of time steps in TMAR, and exactly $\lambda(\psi)$ time steps in EM, these phases terminate constant time. If an agent x is not in a cluster, then x executes a non-empty random walk of length at most $\lambda(\psi) = O(1)$ according to TMAR, hence, r terminates in constant time. □

Until now, we have considered DMAR only on a per-round basis, but it remains to show that the number of rounds before an instance of UMVRP-L is solved is also bounded. The following analysis demonstrates this and will rely on Theorem 3.3 and a known

result about random walks. The *cover time* of a random walk on a connected, undirected graph $G(V, E)$ (denoted $C(G)$) represents the worst-case expected time to visit every node of G starting at any initial node. It is known that $C(G) \leq 2|E|(|V| - 1)$ [29]. We now show that the number of time steps before DMAR solves any solvable instance of UMVRP-L is bounded in expectation.

Theorem 3.4. *For a solvable instance $G(V, E), O, S, \tau$ on N nodes, DMAR completes all tasks in $O(N^2)$ time steps in expectation.*

Proof. During a round of DMAR, if there are clusters in the TMAR phase, then some task will be completed in constant time by Theorem 3.3 since the multiagent rollout algorithm is guaranteed to complete all tasks with our chosen heuristic [10]. Otherwise, each agent follows a random walk of length at least one. In the worst case, an agent may need to visit every node in the network to locate a task via random exploration. Consequently, by Theorem 3.3, the cover time bound, and since $|E| \leq 4N$, we conclude that the total number of time steps required to solve the instance is at most directly proportional to $8N(N-1) = O(N^2)$ in expectation. □

Although we have bounded the number of time steps required for DMAR to solve an instance of UMVRP-L in expectation, it is critical for any motion planner to show that it is *probabilistically complete*, i.e. for any solvable instance I of UMVRP-L, the probability that DMAR solves I tends to unity as the number of rounds tends to infinity [7]. The following theorem demonstrates this.

Theorem 3.5. *DMAR is a probabilisitically complete planner.*

Proof. We argue from the squeeze principle. Let random variable T_{DMAR} be the length of an execution of DMAR in time steps given an arbitrary solvable instance of UMVRP-L. As $\mathbb{E}[T_{DMAR}]$ is finite and positive by Theorem 3.4, the sequence $(\mathbb{E}[T_{DMAR}]/t)_{t=1}^{\infty}$ converges to zero, as does the sequence $(0)_{t=1}^{\infty}$. We use these facts to show the sequence $(\Pr[T_{DMAR} \geq t])_{t=1}^{\infty}$ also converges to zero. Let $\epsilon > 0$ be an arbitrary positive real number. It suffices to show that there exits some $t' \in \mathbb{Z}^+$ such that if $t \geq t'$ then $\Pr[T_{DMAR} \geq t] < \epsilon$. Choose t' such that $\mathbb{E}[T_{DMAR}]/t < \epsilon$ for all $t \geq t'$. Such a t' must exist since $(\mathbb{E}[T_{DMAR}]/t)_{t=1}^{\infty}$ is decreasing and tends to zero. By the Markov inequality, $0 \leq \Pr[T_{DMAR} \geq t] \leq \mathbb{E}[T_{DMAR}]/t < \epsilon, \forall t \in \mathbb{Z}_{\geq t'}$, and so $(\Pr[T_{DMAR} \geq t])_{t=1}^{\infty}$ also tends to zero. Hence, under an execution of DMAR the probability that a reachable non-completed task persists tends to zero as the number of time steps tends to infinity, and so DMAR is probabilistically complete. □

4 EXPERIMENTAL RESULTS

We consider eight classes of instances of UMVRP-L; namely $10 \times 10, \ldots, 80 \times 80$ grid graphs. For each class, we uniformly sample ten grids with 20% of the nodes randomly designated as obstacles. For each $\sqrt{N} \times \sqrt{N}$ grid, we consider a range of k-values (recall sensing radius), and three agent-to-task ratios (1:2,1:1, 2:1). Agents and tasks are distributed uniformly at random over each instance for each ratio, and the number of agents is always $\sqrt{N}$. Moreover, we disregard any topology that contains geographically isolated tasks, i.e. where the expected length of a random walk to visit any task is $\omega(N)$. For all simulations, we fixed $\psi = 8$, as preliminary simulations suggested this yields the best results. We run ten independent runs per instance to account for randomness. For each grid size and k-value combination, we report movement costs, wall-clock running times and number of clusters formed averaged over ten runs of ten instances for each of the three agent-to-task ratios.

For each of these combinations we run DMAR and a corresponding base policy (BP) called the *greedy-exploration policy*. Note that the algorithm that generates this policy differs from DMAR only in Line 44 of Algorithm 1, where instead of multiagent rollout, a greedy nearest-neighbor heuristic is used to compute a policy for each agent in K over $\mathcal{M}(K)$. That is, the base policy still involves random exploration and mandates SOAC and LMA, but does not induce coordination. The combinations described above result in approximately 50,000 individual simulations over 5,000 instances. In this work we present results from 40×40, 60×60 and 80×80 grids, but comprehensive results can be found in Appendix-C [38]. Note that these experiments simulate the constraints of a distributed environment, and are not themselves decentralized. Simulator source code can be found in Appendix-F [38].

We observe from Figure 2 (rows 1-3, left) that for each grid on $N \in \{40^2, 60^2, 80^2\}$ nodes there exists a critical radius $k^*(N)$ such that for all $k \geq k^*(N)$, rollout outperforms the greedy-exploration base policy. As k decreases, we see that the performance of rollout degrades gracefully; and as k increases, the total average running time seems to increase exponentially to account for increased online planning, illustrating the trade-off between scalability and solution quality. From Figure 2 (rows 1-3) we see that an effective range of radii exists that contains a special radius for which there are multiple clusters generated and the relative cost improvement is a factor of approximately two. At this special radius we achieve an average wall-clock running time that is a small fraction of the average time corresponding to the largest k values we considered. Note that the average running time for DMAR on an instance at any k value is no larger than the running time of the centralized multiagent rollout algorithm, hence the observed improvement in running time over the largest k values represents a lower bound when comparing DMAR to centralized MVRP. For k values considered beyond the effective range, we only observe a further constant factor relative cost improvement (approximately 3), even when the number of clusters generated approaches one (as in centralized MVRP), and hence only a small benefit is gained at the expense of significantly higher running times. In Figure 2 (bottom), we plot an approximation of $k^*(N)$ as obtained from our samples, and juxtapose several slow-growing functions. We see that $k^*(N)$ is closely approximated by $\log_2^*(N)$. Consequently, we mark our effective range bounds in Figure 2 (rows 1-3) (indicated by the shaded box) as $2\log_2^*(N)$ to $3\log_2^*(N)$ (i.e. 8-12).

Physics-based Simulations and Robotics Experiments. To address the question of whether DMAR is applicable to real-world scenarios, we have created a proof-of-concept implementation that adapts DMAR to continuous space using the Robotarium platform [39]. These simulations capture the dynamics of differential drive-based robots; that is, each robot's movement is generated by actuation of motors, the dynamics of which are described by a set of ordinary differential equations. In this adaptation, planning by each robot occurs in discrete space (9×9 grids), however, once a discrete control sequence is computed, each control is then mapped to a continuous space velocity vector, after which the new robot

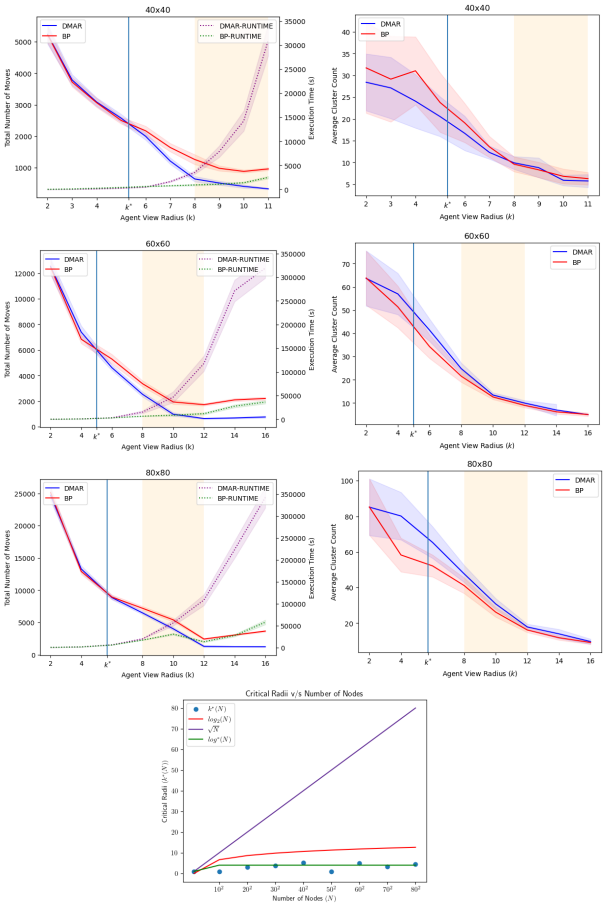

Figure 2: (Rows 1-3, left) Left vertical axes show the average cost of greedy-exploration base policy vs average cost of DMAR. Right vertical axes show average running time in seconds for DMAR and base policy. Critical radii are marked as k^*; shaded orange boxes show effective ranges. (Rows 1-3, right) Show average number of clusters from base policy vs those from DMAR. 95% confidence intervals are shown by shaded regions around respective means. (Bottom) Sampled critical radii function $k^*(N)$.

position is realized by the actuators. We took measures to avoid collisions and simulated the distributed constraints of our model on a centralized server (see Appendix-D [38]).

Due to the size constraints on the arena, we limit our simulations to 30 randomly generated environments of size 2 m×2 m, each with a varying number of tasks and 7 agents. For each instance, we ran 10 executions and obtained average costs. In addition, we also varied the sensing radius k as 20, 40, 60, 80 cm and ∞. For all simulations, we fixed $\psi = 4$. The results of these experiments are presented in Figure 3 (left). Here DMAR outperforms the greedy-exploration policy regardless of k, although its performance degrades as k decreases. We see that as the radius increases, the number of clusters decreases. We also ran experiments on physical robots using the Robotarium platform on many instances (Figure 3 (right)).

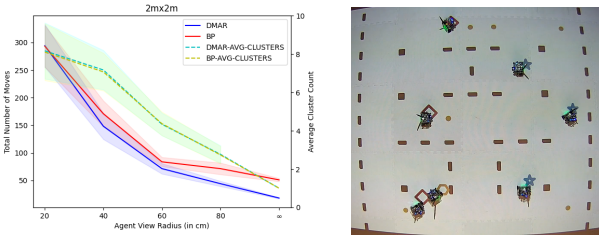

Figure 3: (Left) Physics-based simulation on 2 m×2 m environments. Left vertical axis shows average solution cost, right axis shows average number of clusters. 95% confidence intervals are given by shaded regions. (Right) DMAR execution in Robotarium. Small dots are tasks, boxes are obstacles. Star, diamond, hexagon indicate cluster membership.

5 DISCUSSION

We have presented a distributed computing approach for applying rollout to a generalization of the multivehicle routing problem where agents operate with limited local sensing capabilities and possess no a priori knowledge of the network. This study allowed us to examine the role of communication in rollout, and the effect that limiting it has on performance. Our approach produces quality solutions already for small local sensing radii. We have shown that there exists an (effectively constant) critical sensing radius beyond which rollout outperforms a greedy strategy for all larger radii. Moreover, there exists a range of effective radii where the relative cost improvement over the base policy is approximately a factor of two, but communication is still significantly limited. In this effective range we also observed radii where the running time of DMAR was exponentially smaller with only a constant factor performance detriment as compared to larger radii. This implies the superior scalability of our approach, and illustrates its trade-off with performance. We conclude that the user may choose a radius between 8 and 12 and expect substantial relative cost improvement and fast execution without knowledge of the network size, and that our approach is conceptually adaptable to continuous space and robust to sensor noise, augmenting its real-world applicability.

Regarding the limitations of this work, real-world distributed systems rarely behave synchronously, which we do not address here. Moreover, arbitrary physical environments may be noisy or change over time. As such, future work includes adapting DMAR to stochastic and dynamic environments, and to weaker distributed models. Nevertheless, our approach may apply more broadly to other such decentralized multiagent problems, as vehicle routing is a fundamental primitive of many multiagent algorithms, e.g. [5, 18, 26, 27, 32]. Lastly, our target applications (recall minefield disarmament and post-disaster search and rescue) suggest that DMAR may produce a considerable positive societal impact.

ACKNOWLEDGMENTS

Special thanks to Ted Pavlic for guidance, and to Anya Chaturvedi and Joseph Briones for technical and organizational support. Research partially supported under NSF CCF-1637393 and CCF-1733680 and DoD-ARO MURI No.W911NF-19-1-0233 awards.

REFERENCES

[1] Christopher Amato, George Konidaris, Ariel Anders, Gabriel Cruz, Jonathan P How, and Leslie P Kaelbling. 2016. Policy Search for Multi-Robot Coordination under Uncertainty. *Int. J. Rob. Res.* 35, 14 (dec 2016), 1760–1778.

[2] Christopher Amato, George Konidaris, Gabriel Cruz, Christopher A. Maynor, Jonathan P. How, and Leslie P. Kaelbling. 2015. Planning for decentralized control of multiple robots under uncertainty. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 1241–1248.

[3] Chris Amato, George Dimitri Konidaris, Leslie Pack Kaelbling, and Jonathan P. How. 2019. Modeling and Planning with Macro-Actions in Decentralized POMDPs. *The journal of artificial intelligence research* 64 (2019), 817–859.

[4] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. 2004. Computation in Networks of Passively Mobile Finite-State Sensors. In *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing* (St. John's, Newfoundland, Canada) *(PODC '04)*. Association for Computing Machinery, New York, NY, USA, 290–299. https://doi.org/10.1145/1011767.1011810

[5] G Balamurugan, J Valarmathi, and V P S Naidu. 2016. Survey on UAV navigation in GPS denied environments. In *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*. 198–204. https://doi.org/10.1109/SCOPES.2016.7955787

[6] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum Learning. In *Proceedings of the 26th Annual International Conference on Machine Learning* (Montreal, Quebec, Canada) *(ICML '09)*. Association for Computing Machinery, New York, NY, USA, 41–48. https://doi.org/10.1145/1553374.1553380

[7] Dmitry Berenson and Siddhartha S. Srinivasaz. 2010. Probabilistically complete planning with end-effector pose constraints. In *2010 IEEE International Conference on Robotics and Automation*. 2724–2730. https://doi.org/10.1109/ROBOT.2010.5509694

[8] Daniel S. Bernstein, Shlomo Zilberstein, and Neil Immerman. 2000. The Complexity of Decentralized Control of Markov Decision Processes. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence* (Stanford, California) *(UAI'00)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 32–37.

[9] Dimitri Bertsekas. 2022. Newton's method for reinforcement learning and model predictive control. *Results in Control and Optimization* 7 (June 2022). https://doi.org/10.1016/j.rico.2022.100121 Publisher Copyright: © 2022.

[10] Dimitri P. Bertsekas. 2020. *Rollout, Policy Iteration, and Distributed Reinforcement Learning* (1 ed.). Athena Scientific, Belmont, Massachusetts.

[11] Dimitri P. Bertsekas. 2022. Lessons from AlphaZero for Optimal, Model Predictive, and Adaptive Control. *ArXiv* abs/2108.10315 (2022). https://api.semanticscholar.org/CorpusID:237279336

[12] Dimitri P. Bertsekas and John N. Tsitsiklis. 1996. *Neuro-Dynamic Programming* (1st ed.). Athena Scientific.

[13] Sushmita Bhattacharya, Siva Kailas, Sahil Badyal, Stephanie Gil, and Dimitri P. Bertsekas. 2020. Multiagent rollout and policy iteration for POMDP with application to multi-robot repair problems. In *4th Conference on Robot Learning*. Cambridge, Massachusetts.

[14] Kris Braekers, Katrien Ramaekers, and Inneke Van Nieuwenhuyse. 2016. The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering* 99 (2016), 300–313. https://doi.org/10.1016/j.cie.2015.12.007

[15] G.S. Chirikjian. 1994. Kinematics of a metamorphic robotic system. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. 449–455 vol.1.

[16] Floriano De Rango, Nunzia Palmieri, Xin She Yang, and Salvatore Marano. 2015. Bio-inspired exploring and recruiting tasks in a team of distributed robots over mined regions. In *2015 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*. 1–8. https://doi.org/10.1109/SPECTS.2015.7285279

[17] Zahra Derakhshandeh, Shlomi Dolev, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. 2014. Amoebot - a New Model for Programmable Matter. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures* (Prague, Czech Republic) *(SPAA '14)*. Association for Computing Machinery, New York, NY, USA, 220–222.

[18] Michael Drexl. 2013. Applications of the vehicle routing problem with trailers and transshipments. *European Journal of Operational Research* 227, 2 (2013), 275–283. https://doi.org/10.1016/j.ejor.2012.12.015

[19] Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. 2018. Counterfactual Multi-Agent Policy Gradients. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence* (New Orleans, Louisiana, USA) *(AAAI'18/IAAI'18/EAAI'18)*. AAAI Press, Article 363, 9 pages.

[20] M. R. Garey, D. S. Johnson, and R. Endre Tarjan. 1976. The Planar Hamiltonian Circuit Problem is NP-Complete. *SIAM J. Comput.* 5, 4 (1976), 704–714.

[21] Bruce L. Golden, Attila A. Kovacs, and Edward A. Wasil. [n.d.]. Chapter 14: Vehicle Routing Applications in Disaster Relief. In *Vehicle Routing*. Society for Industrial and Applied Mathematics, 409–436. https://doi.org/10.1137/1.9781611973594.ch14

[22] Jayesh K. Gupta, Maxim Egorov, and Mykel Kochenderfer. 2017. Cooperative Multi-agent Control Using Deep Reinforcement Learning. In *Autonomous Agents and Multiagent Systems*, Gita Sukthankar and Juan A. Rodriguez-Aguilar (Eds.). Springer International Publishing, Cham, 66–83.

[23] Ming Han and Yabin Wang. 2018. A Survey for Vehicle Routing Problems and Its Derivatives. *IOP Conference Series: Materials Science and Engineering* 452, 4 (dec 2018), 042024. https://doi.org/10.1088/1757-899X/452/4/042024

[24] Stefan Irnich, Paolo Toth, and Daniele Vigo. 2014. *Chapter 1: The Family of Vehicle Routing Problems*. 1–33. https://doi.org/10.1137/1.9781611973594.ch1 arXiv:https://epubs.siam.org/doi/pdf/10.1137/1.9781611973594.ch1

[25] Soummya Kar, José M. F. Moura, and H. Vincent Poor. 2013. $\mathcal{QD}$-Learning: A Collaborative Distributed Strategy for Multi-Agent Reinforcement Learning Through Consensus + Innovations. *IEEE Transactions on Signal Processing* 61, 7 (2013), 1848–1862.

[26] Veronika Lesch, Martin Breitbach, Michele Segata, Christian Becker, Samuel Kounev, and Christian Krupitzer. 2022. An Overview on Approaches for Coordination of Platoons. *IEEE Transactions on Intelligent Transportation Systems* 23, 8 (2022), 10049–10065. https://doi.org/10.1109/TITS.2021.3115908

[27] Shengbo Eben Li, Yang Zheng, Keqiang Li, and Jianqiang Wang. 2015. An overview of vehicular platoon control under the four-component framework. In *2015 IEEE Intelligent Vehicles Symposium (IV)*. 286–291. https://doi.org/10.1109/IVS.2015.7225700

[28] Fei Liu, Chengyu Lu, Lin Gui, Qingfu Zhang, Xialiang Tong, and Mingxuan Yuan. 2023. Heuristics for Vehicle Routing Problem: A Survey and Recent Advances. arXiv:2303.04147 [cs.AI]

[29] Rajeev Motwani and Prabhakar Raghavan. 1995. *Randomized Algorithms*. Cambridge University Press. https://doi.org/10.1017/CBO9780511814075

[30] Dheeraj Nagaraj, Naman Agarwal, Praneeth Netrapalli, Prateek Jain, and Syomantak Chaudhari (Eds.). 2022. *Online Target Q-learning with Reverse Experience Replay: Efficiently finding the Optimal Policy for Linear MDPs*.

[31] Harilaos N. Psaraftis, Min Wen, and Christos A. Kontovas. 2016. Dynamic vehicle routing problems: Three decades and counting. *Networks* 67, 1 (2016), 3–31. https://doi.org/10.1002/net.21628 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/net.21628

[32] Alberto Quattrini Li. 2020. Exploration and Mapping with Groups of Robots: Recent Trends. *Current Robotics Reports* 1 (12 2020), 1–11. https://doi.org/10.1007/s43154-020-00030-5

[33] Syed Mohib Raza, Mohammad Sajid, and Jagendra Singh. 2022. Vehicle Routing Problem Using Reinforcement Learning: Recent Advancements. In *Advanced Machine Intelligence and Signal Processing*, Deepak Gupta, Koj Sambyo, Mukesh Prasad, and Sonali Agarwal (Eds.). Springer Nature Singapore, Singapore, 269–280.

[34] Stuart Russell and Peter Norvig. 2009. *Artificial Intelligence: A Modern Approach* (3rd ed.). Prentice Hall Press, USA.

[35] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[36] Paolo Toth and Daniele Vigo. 2002. *The Vehicle Routing Problem. Monographs on Discrete Mathematics and Applications*. Vol. 9. Philadelphia: Society for Industrial and Applied Mathematics.

[37] J.E. Walter, E.M. Tsai, and N.M. Amato. 2005. Algorithms for fast concurrent reconfiguration of hexagonal metamorphic robots. *IEEE Transactions on Robotics* 21, 4 (2005), 621–631.

[38] Jamison W. Weber, Dhanush R. Giriyan, Devendra R. Parkar, Andréa W. Richa, and Dimitri P. Bertsekas. 2023. Distributed Online Rollout for Multivehicle Routing in Unmapped Environments. arXiv:2305.15596 [cs.DC]

[39] Sean Wilson, Paul Glotfelter, Siddharth Mayya, Gennaro Notomista, Yousef Emam, Xiaoyi Cai, and Magnus Egerstedt. 2021. The Robotarium: Automation of a Remotely Accessible, Multi-Robot Testbed. *IEEE Robotics and Automation Letters* 6, 2 (2021), 2922–2929.

[40] Damien Woods, Ho-Lin Chen, Scott Goodfriend, Nadine Dabby, Erik Winfree, and Peng Yin. 2013. Active Self-Assembly of Algorithmic Shapes and Patterns in Polylogarithmic Time. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science* (Berkeley, California, USA) *(ITCS '13)*. Association for Computing Machinery, New York, NY, USA, 353–354. https://doi.org/10.1145/2422436.2422476

[41] Mark Yim, Wei-min Shen, Behnam Salemi, Daniela Rus, Mark Moll, Hod Lipson, Eric Klavins, and Gregory S. Chirikjian. 2007. Modular Self-Reconfigurable Robot Systems [Grand Challenges of Robotics]. *IEEE Robotics and Automation Magazine* 14, 1 (2007), 43–52.