Monotonic Target Assignment for Robotic Networks

Stephen L. Smith Francesco Bullo

Abstract

Consider an equal number of mobile robotic agents and distinct target locations dispersed in an environment. Each agent has a limited communication range and either (1) knowledge of every target position, or (2) a finite-range sensor capable of acquiring target positions and no a priori knowledge of target positions. In this paper we study the following target assignment problem: design a distributed algorithm with which the agents divide the targets among themselves and, simultaneously, move to their unique target. We evaluate an algorithm's performance by characterizing its worst-case asymptotic time to complete the target assignment; that is the task completion time as the number of agents (and targets) increases, and the size of the environment scales to accommodate them. We introduce the intuitive class of *monotonic algorithms*, and give a lower bound on its worst-case completion time. We design and analyze two algorithms within this class: the ETSP ASSGMT algorithm which works under assumption (1), and the GRID ASSGMT algorithm which works under either assumption (1) or (2). In "sparse environments," where communication is infrequent, the ETSP ASSGMT algorithm is within a constant factor of the optimal monotonic algorithm for worst-case initial conditions. In "dense environments," where communication is more prevalent, the GRID ASSGMT algorithm is within a constant factor of the optimal monotonic algorithm for worst-case initial conditions. In addition we characterize the performance of the GRID ASSGMT algorithm for uniformly distributed targets and agents, and for the case when there are more agents than targets.

I. INTRODUCTION

Consider a group of n mobile robotic agents, equipped with wireless transceivers for limited range communication, dispersed in an environment $\mathcal{E} \subset \mathbb{R}^2$ which contains n target locations. In addition, consider two scenarios: (1) each agent is given a list containing all target positions (the positions may be given as GPS coordinates); or (2) each agent has no initial target information, but has a finite-range target sensor to acquire target positions. The task is for the agents to divide the targets among themselves so that in minimum time, each target location is occupied by an agent. Since no *a priori* assignment of target-agent pairs has been given, the agents must solve the problem through communication and motion. We call this the *target assignment problem*. This problem has many applications in

Preliminary versions of this work appeared as [1] and as [2]. This material is based upon work supported in part by ARO MURI Award W911NF-05-1-0219 and ONR Award N00014-07-1-0721.

Stephen L. Smith and Francesco Bullo are with Department of Mechanical Engineering and the Center for Control, Dynamical Systems and Computation, University of California, Santa Barbara, CA 93106-5070, USA. E-mail: stephen@engineering.ucsb.edu, bullo@engineering.ucsb.edu

UAV surveillance and exploration, or mobile sensor networks. The first scenario could arise when a high-altitude, sensory-rich aircraft communicates a large number of target positions to a correspondingly large group of smaller, slower, autonomous aircraft at lower altitudes. The second (local sensing) scenario could arise in exploration tasks where a group of UAVs are sent into a region to find, and provide service to, spatially distributed tasks.

The centralized problem of simply assigning one agent to each target is known in the combinatorial optimization literature as the *maximum matching problem* [3]. To efficiently assign agents to targets, we may be interested in finding a maximum matching (i.e., an assignment of one agent to each target) that minimizes a cost function. If the cost function is the sum of distances from each agent to its assigned target, then the problem is known as the *sum assignment problem*, or the *minimum weight maximum matching problem*, [3]. Another choice of cost function is to minimize the maximum distance between agents and their assigned targets. This problem is commonly referred to as the *bottleneck assignment problem* [4]. There exist efficient polynomial time algorithms for the solution of all these problems [5], [6], [4]. Additionally, the sum assignment problem can be solved in a parallel fashion via the *auction algorithm* [7]. However, these solutions do not directly apply to our problem where, due to the agents' limited communication range, the communication topology is time-varying, and possibly disconnected.

The class of problems commonly referred to as *decentralized task allocation* for UAVs (or UGVs), is closely related to our target assignment problem. In these problems the goal is generally to assign vehicles to spatially distributed tasks while maximizing the "score" of the mission. In [8] a taxonomy of task allocation problems is given, dividing problems into groups based on, among other things, the number of tasks a robot can execute, and the number of robots required for a task. In papers such as [9], [10], [11], advanced heuristic methods are developed, and their effectiveness is demonstrated through simulation or real world implementation. In [12] the auction algorithm is adapted to solve a task allocation problem in the presence of communication delays. In [13] the authors study the problem of dynamically reassigning agents as new tasks arrive and old tasks expire. There has also been prior work on target assignment problems [14], [15]. In [14] the authors formulate a target assignment problem as a multi-player game and seek to optimize a global utility. In [15] an algorithm based on hybrid systems tools is developed and its performance is characterized by a bound on the number of switches of the hybrid system. Unlike the prior work, in this paper we study the scalability properties of the minimum-time target assignment problem. We assume that each agent has limited communication capabilities and either (1) *full target knowledge* (i.e., each agent knows the position of every target), or (2) *local target sensing* (i.e., each agent has a finite-range target sensor to acquire target positions). We focus on characterizing the completion time as the number of agents n grows, and the square environment $\mathcal{E}(n)$ grows to accommodate them.[1]

The contributions of this paper are: a novel and concise statement of the minimum-time target assignment problem for robotic networks; a broad class of distributed algorithms for solving this problem; lower bounds on the worst-case performance achievable by any algorithm in this class; and algorithms which perform within a constant factor

[1]The size of the square environment $\mathcal{E}$ is a function of n, and thus we write $\mathcal{E}(n)$. If the environment size were independent of n, then the density of robots would become arbitrarily large as the task size n became large, which is not realistic. Thus, either the environment should grow with n (as is assumed here), or the robot's attributes should shrink with n (as is discussed in Section VIII-C).

of the optimal monotonic algorithm for worst-case initial conditions. In Section IV-A we introduce the class of *monotonic algorithms,* which provides an intuitive approach for solving the target assignment problem. We show that in "sparse environments," that is when $|\mathcal{E}(n)|/n \to +\infty$, for every monotonic algorithm there exists a (worst-case) set of initial target and agent positions such that the completion time is in $\Omega(\sqrt{|\mathcal{E}(n)|n})$.[2] In "dense environments," that is when $|\mathcal{E}(n)|/n \to 0^+$, every algorithm in the class has worst-case completion time in $\Omega(|\mathcal{E}(n)|)$. In Section V, we assume full target knowledge and present a monotonic algorithm, called the ETSP ASSGMT algorithm, with worst-case completion time in $O(\sqrt{|\mathcal{E}(n)|n})$. In this algorithm, each agent computes an ETSP tour through the n targets, turning the cloud of target points into an ordered ring. Agents then move along the ring, looking for the next available target. When agents communicate, they exchange information on the location of the next available target along the ring. Then, in Section VI we present a monotonic algorithm, called the GRID ASSGMT algorithm, which operates under either the full target knowledge assumption, or the local target sensing assumption as long as the sensing range is at least $\sqrt{2/5}$ times the communication range. Under either assumption, the GRID ASSGMT algorithm has worst-case completion time in $O(|\mathcal{E}(n)|)$. In this algorithm, the agents partition the environment into cells, and determine local maximum assignments in the cell which they occupy. A leader is elected in each cell, and through communication between leaders of adjacent cells, local assignments are merged into a global and complete assignment. These two algorithms are complementary in terms of worst-case performance: in "sparse environments," the ETSP ASSGMT algorithm is within a constant factor of the optimal monotonic algorithm, and is "dense environments," the GRID ASSGMT algorithm is within a constant factor of the optimal monotonic algorithm.

We also characterize the stochastic properties of the GRID ASSGMT algorithm in "dense environments." If the agents and targets are uniformly distributed, then the completion time belongs to $O(\sqrt{|\mathcal{E}(n)|})$ with high probability. Additionally, if there are n agents and only $n/\log n$ targets, then the completion time belongs to $O(1)$ with high probability. In Section VIII we discuss extensions of the ETSP ASSGMT and GRID ASSGMT algorithms to higher dimensional spaces and to the case of n agents and m targets, $n \neq m$.

II. COMBINATORIC, GEOMETRIC AND STOCHASTIC PRELIMINARIES

In this section we review a few useful results on the centralized matching problem, the Euclidean traveling salesperson problem, occupancy problems, and random geometric graphs. We let $\mathbb{R}$, $\mathbb{R}_{\geq 0}$ and $\mathbb{N}$ denote the set of real numbers, the set of non-negative real numbers, and the set of positive integers, respectively. Given a finite set A, we let $|A|$ denote its cardinality, and given an infinite set $A \subset \mathbb{R}^2$ we let $|A|$ denote its area. For two functions $f, g : \mathbb{N} \to \mathbb{R}_{>0}$, we write $f(n) \in O(g)$ (respectively, $f(n) \in \Omega(g)$) if there exist $N \in \mathbb{N}$ and $c \in \mathbb{R}_{>0}$ such that $f(n) \leq cg(n)$ for all $n \geq N$ (respectively, $f(n) \geq cg(n)$ for all $n \geq N$). If $f(n) \in O(g)$ and $f(n) \in \Omega(g)$, then we say $f(n) \in \Theta(g)$. We say that event $A(n)$ occurs *with high probability* (w.h.p.) if the probability of $A(n)$ occurring tends to one as $n \to +\infty$.

[2] $|\mathcal{E}(n)|$ denotes the area of $\mathcal{E}(n)$, and $\Omega(\cdot)$ is the asymptotic notation for lower bounds as reviewed in Section II.

A. Centralized Matching

Consider n persons and the problem of dividing them among n tasks. For each person i, there is a nonempty set $\mathcal{Q}^{[i]}$ of tasks that i can be assigned to, and cost $c_{ij} \geq 0$ associated to each task $j \in \mathcal{Q}^{[i]}$. An *assignment* or *matching* M is a set of person-task pairs (i, j) such that $j \in \mathcal{Q}^{[i]}$ for all $(i, j) \in M$, and such that for each person i (likewise, task j) there is at most one pair $(i, j) \in M$. The matching M is a *maximum* matching if for every matching $\tilde{M}$, we have $|\tilde{M}| \leq |M|$. If $|M| = n$, then the matching is *complete*. The matching M is *maximal* if there does not exist a matching $\tilde{M}$, such that $\tilde{M}$ is a strict superset of M. There are several polynomial time algorithms for determining a maximum matching. Weighted maximum matching problems are those of finding the maximum matching M that minimizes a cost function. Two common cost functions are the sum, $\sum_{(i,j)\in M} c_{ij}$, or the bottleneck $\max_{(i,j)\in M} c_{ij}$, and polynomial time algorithms exist for the solution of both of these problems [3].

In this paper we will require a standard algorithm, called MAXIMAL MATCH, for computing a maximal matching. The algorithm chooses the person-task pair with lowest cost, adds it to the matching, removes the person and task from the problem, and repeats. In the case when each person can be assigned to any of the n tasks (i.e., for each person i, the set $\mathcal{Q}^{[i]}$ contains all n tasks), this algorithm determines a complete, and thus maximum, matching.

```
MAXIMAL MATCH, outputs a maximal matching $M$
1 Initialize $M := \emptyset$, and $\mathcal{I}_i := \{1, \dots, n\}$.
2 while there exists an $i \in \mathcal{I}_i$ with $|\mathcal{Q}^{[i]}| \neq 0$ do
3     Compute the indices $(i^*, j^*) := \arg\min_{i\in\mathcal{I}_i, j\in\mathcal{Q}^{[i]}} c_{ij}$
4     Set $M := M \cup (i^*, j^*)$, $\mathcal{I}_i := \mathcal{I}_i \setminus \{i^*\}$, and for each $i \in \mathcal{I}_i$, $\mathcal{Q}^{[i]} := \mathcal{Q}^{[i]} \setminus \{j^*\}$
```

B. The Euclidean Traveling Salesperson Problem

For a set $\mathcal{Q}$ of n points in $\mathbb{R}^2$, let $\mathrm{ETSP}(\mathcal{Q})$ denote the length of the shortest closed path through all points in $\mathcal{Q}$. The following result characterizes the length of this path when $\mathcal{Q} \subset \mathcal{E}(n)$, where (for consistency with the remainder of this paper) $\mathcal{E}(n)$ is a square environment that is compact for each n.

Theorem 2.1 (ETSP tour length, [16]): If $\mathcal{Q}$ is a set of n points in $\mathcal{E}(n)$, then $\mathrm{ETSP}(\mathcal{Q}) \in O(\sqrt{n|\mathcal{E}(n)|})$.

The problem of computing an optimal ETSP tour is known to be NP-complete. However, there exist polynomial time approximation schemes. For example, it is shown in [17] that a tour no longer than $(1+\epsilon)$ times the shortest one can be found in $n(\log n)^{O(1/\epsilon)}$ computation time.

C. Occupancy Problems

Occupancy problems, or "bins and balls" problems, are concerned with randomly distributing m balls into n equally sized bins. The two results we present here will be useful in our analysis.

Theorem 2.2 (Bins and balls properties, [18], [19]): Consider uniformly randomly distributing m balls into n bins and let γ be any function such that $\gamma(n) \to +\infty$ as $n \to +\infty$. The following statements hold:

(i) if $m = n$, then w.h.p. each bin contains $O\left(\frac{\log n}{\log\log n}\right)$ balls;

(ii) if $m = n\log n + \gamma(n)n$, then w.h.p. there exist no empty bins;

(iii) if $m = n\log n - \gamma(n)n$, then w.h.p. there exists an empty bin;

(iv) if $m = Kn\log n$, where $K > 1/\log(4/\mathrm{e})$, then w.h.p. every bin contains $\Theta(\log n)$ balls.

We will be interested in partitioning a square environment into equally sized and openly disjoint square bins such that the area of each bin is "small." To do this, we require the following simple fact.

Lemma 2.3 (Dividing the environment): Given $n \in \mathbb{N}$ and $r_{\text{comm}} > 0$, consider a square environment $\mathcal{E}(n)$. If $\mathcal{E}(n)$ is partitioned into b^2 equally sized and openly disjoint square bins, where

$$b := \left\lceil \frac{\sqrt{5|\mathcal{E}(n)|}}{r_{\text{comm}}} \right\rceil, \tag{1}$$

then the area of each bin is no more than $r_{\text{comm}}^2/5$. Moreover, if $x, y \in \mathcal{E}(n)$ are in the same bin or in adjacent bins, then $\|x - y\| \leq r_{\text{comm}}$.

D. Random Geometric Graphs

For $n \in \mathbb{N}$ and $r_{\text{comm}} \in \mathbb{R}_{>0}$, a planar *geometric graph* $G(n, r_{\text{comm}})$ consists of n vertices in $\mathbb{R}^2$, and undirected edges connecting all vertex pairs $\{x, y\}$ with $\|x - y\| \leq r_{\text{comm}}$. We also refer to this as the r_{comm}-geometric graph. If the vertices are randomly distributed in some subset of $\mathbb{R}^2$, then we call the graph a *random geometric graph*.

Theorem 2.4 (Connectivity of random geometric graphs, [20]): Consider the random geometric graph $G(n, r_{\text{comm}})$ obtained by uniformly randomly distributing n points in the square environment $\mathcal{E}(n)$ with

$$\frac{\pi r_{\text{comm}}^2}{|\mathcal{E}(n)|} = \frac{\log n + \gamma(n)}{n}.$$

Then $G(n, r_{\text{comm}})$ is connected w.h.p. if and only if $\gamma(n) \to +\infty$ as $n \to +\infty$.

This theorem will be important for understanding some of our results, as it provides a bound on the environment size necessary for the communication graph of n randomly deployed agents to be asymptotically connected.

III. Network Model and Problem Statement

In this section we formalize our agent and target models and define the sparse and dense environments.

A. Robotic Network Model

Consider n agents in an environment $\mathcal{E}(n) := [0, \ell(n)]^2 \subset \mathbb{R}^2$, where $\ell(n) > 0$ (that is, $\mathcal{E}(n)$ is a square with side length $\ell(n)$). The environment $\mathcal{E}(n)$ is compact for each n but its size depends on n. A robotic agent, $\mathcal{A}^{[i]}$, $i \in \mathcal{I} := \{1, \ldots, n\}$, is described by the tuple

$$\mathcal{A}^{[i]} := \{\text{UID}^{[i]}, \mathbf{p}^{[i]}, r_{\text{comm}}, r_{\text{sense}}\mathbf{u}^{[i]}, M^{[i]}\},$$

where the quantities are as follows: Its unique identifier (UID) is $\text{UID}^{[i]}$, taken from the set $I_{\text{UID}} \subset \mathbb{N}$. Note that each agent does not know the set of UIDs being used and thus does not initially know the magnitude of its UID relative to those of other agents. Its position is $\mathbf{p}^{[i]} \in \mathcal{E}(n)$. Its communication range is $r_{\text{comm}} > 0$, i.e., two agents, $\mathcal{A}^{[i]}$ and $\mathcal{A}^{[k]}$, $i, k \in \mathcal{I}$, can communicate if and only if $\|\mathbf{p}^{[i]} - \mathbf{p}^{[k]}\| \leq r_{\text{comm}}$. Its target sensing range is r_{sense}. With this sensor agent i can determine the relative position of targets within distance r_{sense} of $\mathbf{p}^{[i]}$. Its continuous time velocity input is $\mathbf{u}^{[i]}$, corresponding to the kinematic model $\dot{\mathbf{p}}^{[i]} = \mathbf{u}^{[i]}$, where $\|\mathbf{u}^{[i]}\| \leq v_{\max}$ for some $v_{\max} > 0$. Finally, its memory is $M^{[i]}$ and is of cardinality (size) $|M^{[i]}|$. From now on, we simply refer to agent $\mathcal{A}^{[i]}$ as agent i.

The agents move in continuous time and communicate according to a synchronous discrete time schedule consisting of an increasing sequence $\{t_k\}_{k\in\mathbb{N}}$ of time instants with no accumulation points. We assume $|t_{k+1} - t_k| \leq t_{\max}$, for all $k \in \mathbb{N}$, where $t_{\max} \in \mathbb{R}_{>0}$. We also assume that the time between communication rounds $t_{\max}$ is much smaller than $r_{\text{comm}}/v_{\max}$, the amount of time taken to travel the distance r_{comm}. At each communication round, agents can exchange messages of length $O(\log n)$.[3] Communication round k occurs at time t_k, and all messages are sent and received instantaneously at t_k. Motion then occurs from t_k until t_{k+1}. It should be noted that in this setup we are emphasizing the time complexity due to the motion of the agents.

B. The Target Assignment Problem

Let $\mathcal{Q} := \{\mathbf{q}_1, \dots, \mathbf{q}_n\} \subset \mathcal{E}(n)$ be a set of distinct target locations. In this paper we make one of two assumptions:

Full target knowledge: Each agent knows the position of every target. Thus, agent i's memory, $M^{[i]}$, contains a copy of $\mathcal{Q}$, which we denote $\mathcal{Q}^{[i]}$. To store $\mathcal{Q}^{[i]}$ the size of each agents' memory, $|M^{[i]}|$, must be in $\Omega(n)$.

Local target sensing: Each agent has no initial target information (i.e., $\mathcal{Q}^{[i]} = \emptyset$), but can acquire target positions through its target sensor of range r_{sense}.

Our goal is to solve the *target assignment problem*:

> Determine an algorithm for $n \in \mathbb{N}$ agents, with attributes as described above, satisfying the following requirement; there exists a time $T \geq 0$ such that for each target $\mathbf{q}_j \in \mathcal{Q}$, there is a unique agent $i \in \mathcal{I}$, with $\mathbf{p}^{[i]}(t) = \mathbf{q}_j$ for all $t \geq T$.

If the task begins at time $t = 0$, then the *completion time* of the target assignment task is the minimum $T \geq 0$, such that for each $\mathbf{q}_j \in \mathcal{Q}$, there is a unique $i \in \mathcal{I}$, with $\mathbf{p}^{[i]}(t) = \mathbf{q}_j$ for all $t \geq T$. In this paper we seek algorithms that minimize this completion time. Note that in the local target sensing assumption the agents have less target information than in the full target knowledge assumption. Because of this, an algorithm's performance under the local target sensing assumption can be no better than its performance under the full target knowledge assumption.

Remark 3.1 (Consistent target knowledge): Another possible assumption on the target sets, $\mathcal{Q}^{[i]}$, which still ensures the existence of a complete matching, is the *consistent target knowledge* assumption: For each $\mathcal{K} \subseteq \mathcal{I}$, $\left|\cup_{k\in\mathcal{K}} \mathcal{Q}^{[k]}\right| \geq |\mathcal{K}|$. In fact, it was proved by Frobenius in 1917 and by Hall in 1935 that this is the necessary and sufficient condition for the existence of a complete matching [3]. •

[3]The number of bits required to represent an ID, unique among n agents, is directly proportional to the logarithm of n.

C. Sparse, Dense, and Critical Environments

We wish to study the scalability of a particular approach to the target assignment problem; that is, how the completion time increases as we increase the number of agents, n. The velocity v_{max} and communication range r_{comm} of each agent are independent of n. However, we assume that the size of the environment increases with n in order to accommodate an increase in agents. Borrowing terms from the random geometric graph literature [20], we say that the environment is *sparse* if, as we increase the number of agents, the environment grows quickly enough that the density of agents (as measured by the sum of their communication footprints) decreases; we say the environment is *critical*, if the density is constant, and we say the environment is *dense* if the density increases. Formally, we have the following definition.

Definition 3.2 (Dense, critical and sparse environments): The environment $\mathcal{E}(n)$ is

(i) *sparse* if $|\mathcal{E}(n)|/n \to +\infty$ as $n \to +\infty$;

(ii) *critical* if $|\mathcal{E}(n)|/n \to$ const $\in \mathbb{R}_{>0}$ as $n \to +\infty$;

(iii) *dense* if $|\mathcal{E}(n)|/n \to 0^+$, as $n \to +\infty$.

It should be emphasized that a dense environment does not imply that the communication graph between agents is dense. On the contrary, from Theorem 2.4 we see that the communication graph at random agent positions in a dense environment may not even be connected.

IV. Classes of Algorithms

In this section we introduce a class of algorithms for the target assignment problem that provides the structure for algorithms developed in this paper. We will provide a lower bound on the classes performance using the full target knowledge assumption. Necessarily this also provides a lower bound for the problem using the local target sensing assumption.

A. Monotonic Algorithms

We introduce a class of algorithms which provides an intuitive approach to target assignment.

Definition 4.1 (Monotonic algorithms): A target assignment algorithm is *monotonic* if it is deterministic and has the following property: If a subset of agents $\mathcal{J} \subset \mathcal{I}$ are all located at target $\mathbf{q}_j$ at time t_1 (i.e., $\mathbf{p}^{[i]}(t_1) = \mathbf{q}_j$, $\forall\, i \in \mathcal{J}$), then at least one agent in $\mathcal{J}$ remains located at $\mathbf{q}_j$ for all $t > t_1$ (i.e., $\exists\, i \in \mathcal{J}$ such that $\mathbf{p}^{[i]}(t) = \mathbf{q}_j$, $\forall\, t > t_1$).

We call these algorithms "monotonic" since occupied targets remain occupied for all time, and thus the number of occupied targets monotonically increases throughout the execution. We focus on monotonic algorithms for two reasons: First, monotonicity is natural constraint for target assignment problems since in many scenarios it the agents will begin servicing a target immediately upon arriving at its location—in non-monotonic algorithms, service will be halted as agents leave their targets. Second, monotonic algorithms provide a broad class of algorithms for which rigorous analysis remains tractable.

We are now ready to lower bound the worst-case asymptotic completion time of the target assignment problem for any monotonic algorithm. This bound holds under both the full target knowledge and local target sensing assumptions.

Theorem 4.2 (Time complexity of target assignment): Consider n agents, with communication range $r_{\text{comm}} > 0$, and n targets in $\mathcal{E}(n)$. For all monotonic algorithms the worst-case completion time of the target assignment problem is lower bounded as follows:

(i) if $\mathcal{E}(n)$ is sparse, then the completion time is in $\Omega(\sqrt{n|\mathcal{E}(n)|})$;
(ii) if $\mathcal{E}(n)$ is critical, then the completion time is in $\Omega(n)$;
(iii) if $\mathcal{E}(n)$ is dense, then the completion time is in $\Omega(|\mathcal{E}(n)|)$.

Proof: The proof proceeds by constructing a set of agent and target positions such that the lower bound is achieved. To do this, we place the targets in $\mathcal{E}(n)$ such that the r_{comm}-geometric graph, generated by the target positions, has a maximum number of disconnected components. Next we place agents $2, \dots, n$ so that they occupy targets $\mathbf{q}_2, \dots, \mathbf{q}_n$. We then place agent 1 in $\mathcal{E}(n) \setminus \mathcal{Q}$. If the agents run a monotonic algorithm to solve the target assignment problem, then agents $2, \dots, n$ will not move, and thus the assignment will not be complete until agent 1 reaches target $\mathbf{q}_1$. In the best case, when agent 1 comes within distance r_{comm} of a connected component, it immediately determines whether or not there is a free target in that component (i.e., whether or not $\mathbf{q}_1$ is in that component). However, agent 1 will not receive information about the availability of any targets outside of that component. So, agent 1 must come within distance r_{comm} of the connected component containing $\mathbf{q}_1$, before the assignment can be completed. Since the algorithm is deterministic, we can place the targets, and agents such that the connected component containing $\mathbf{q}_1$ is the last connected component that agent 1 will visit.

To create the maximum number of disconnected components, we partition the environment, $\mathcal{E}(n)$ into P equally sized, and openly disjoint squares, as shown in Fig. 1(a). We consider two cases, based on whether or not there exists an $\epsilon > 0$ such that $|\mathcal{E}(n)| \geq (2r_{\text{comm}} + \epsilon)^2 n$.

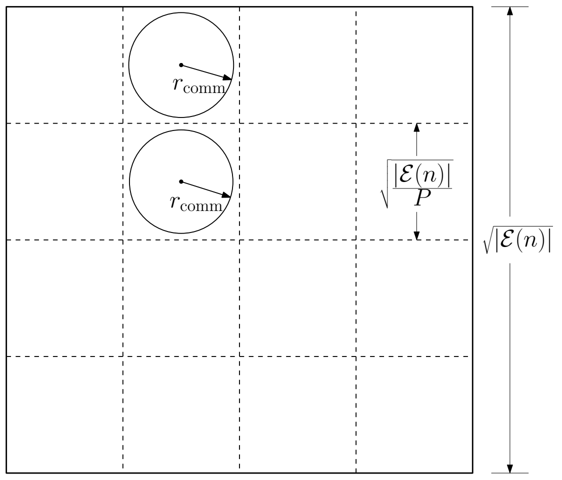
Case 1: [there exists $\epsilon > 0$ such that $|\mathcal{E}(n)| \geq (2r_{\text{comm}} + \epsilon)^2 n$] In this case we set $P := \lceil \sqrt{n} \rceil^2$ and place a target at the center of each square until there are no targets remaining. The area of each square is given by $|\mathcal{E}(n)|/P$, and thus the distance between any two targets is lower bounded by $\sqrt{|\mathcal{E}(n)|/P} \geq \sqrt{(2r_{\text{comm}} + \epsilon)^2 n / \lceil \sqrt{n} \rceil^2}$, which for sufficiently large n, is greater than $2r_{\text{comm}}$. Thus, we have created n disconnected components, as depicted in Fig. 1(a). The distance between r_{comm}-disks centered at any two targets is lower bounded by $\sqrt{|\mathcal{E}(n)|/\lceil \sqrt{n} \rceil^2} - 2r_{\text{comm}}$, and we can place the agents and targets such that one agent must travel this distance $n-1$ times. Thus, the worst-case travel distance is lower bounded by

$$(n-1)\left(\sqrt{\frac{|\mathcal{E}(n)|}{\lceil \sqrt{n} \rceil^2}} - 2r_{\text{comm}}\right) \in \Omega(\sqrt{|\mathcal{E}(n)|n}).$$

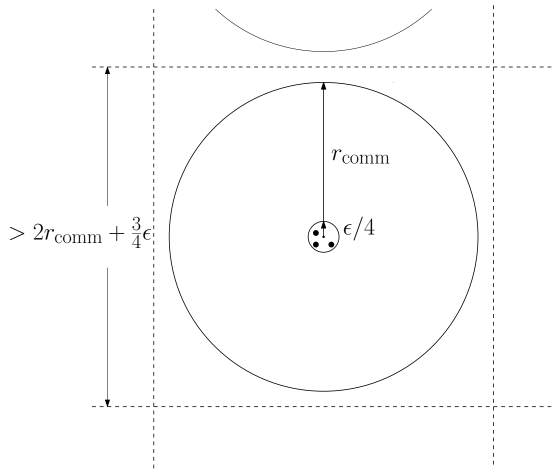
Since the robots travel at constant speed, the completion time is also in $\Omega(\sqrt{|\mathcal{E}(n)|n})$.

Case 2: [for every $\epsilon > 0$, $|\mathcal{E}(n)| < (2r_{\text{comm}} + \epsilon)^2 n$] In this case we fix any $\epsilon > 0$ and set

$$P := \left\lceil \sqrt{\frac{|\mathcal{E}(n)|}{(2r_{\text{comm}} + \epsilon)^2}} \right\rceil^2 .$$


r_{comm}
r_{comm}
$\sqrt{\frac{|\mathcal{E}(n)|}{P}}$
$\sqrt{|\mathcal{E}(n)|}$

(a) Partitioning the environment to construct target positions that generate an r_{comm}-geometric graph with a maximum number of disconnected components.



(b) The $\epsilon/4$-disk located at the center of one of the squares in the partition. Targets are shown in the disk, along with a lower bound on the size of the square.

Fig. 1. Partitioning the environment $\mathcal{E}(n)$ for the proof of Theorem 4.2.

We define a disk of radius $\epsilon/4$ at the center of each of the P squares. We then place $\lceil n/P \rceil$ targets in each $\epsilon/4$-disk, until there are no targets remaining, as shown in Fig. 1(b). Note that for any $\alpha \in]0,1[$, we can find $N \in \mathbb{N}$ such that

$$P < \frac{|\mathcal{E}(n)|}{(2r_{\text{comm}} + \alpha\epsilon)^2}, \quad \text{for all } n \geq N.$$

Letting $\alpha = 3/4$, we find that for large n, the distance between the centers of any two squares is lower bounded by $\sqrt{\frac{|\mathcal{E}(n)|}{P}} \geq 2r_{\text{comm}} + 3\epsilon/4$, as shown in Fig. 1(b). So, the distance between any two $\epsilon/4$-disks is lower bounded by $2r_{\text{comm}} + \epsilon/4$. Thus, we have created $\Omega(P)$ disconnected components. The distance between r_{comm}-disks centered at any two targets in different squares is lower bounded by $\epsilon/4$. Again, we can place the agents and targets such that one agent will have to travel this distance $\Omega(P)$ times. Thus, the worst-case distance is lower bounded by

$$\frac{\epsilon}{4}\Omega(P) \in \Omega(|\mathcal{E}(n)|).$$

Since the robots travel at constant speed, the completion time is also in $\Omega(|\mathcal{E}(n)|)$.

Thus, if $|\mathcal{E}(n)|/n \to +\infty$ as $n \to +\infty$, then we are in Case 1 and the completion time is in $\Omega(\sqrt{|\mathcal{E}(n)|n})$. If $|\mathcal{E}(n)|/n \to \text{const} \in \mathbb{R}_{>0}$ as $n \to +\infty$, then we may be in either Case 1 or Case 2, depending on the value of const, but in either case the completion time is in $\Omega(n)$. Finally, if $|\mathcal{E}(n)|/n \to 0^+$ as $n \to +\infty$, then we are in Case 2 and the completion time is in $\Omega(|\mathcal{E}(n)|)$. ■

Remark 4.3 (Interpretation of lower bound): In Theorem 4.2 we provided a worst-case lower bound. This should be interpreted as follows. For every monotonic algorithm there exists a set of initial target and agent position for which the completion time is no smaller than the lower bound. It should be noted that there are many initial positions for which the completion time is less than this worst-case lower bound (indeed, there are initial positions for which the completion time is zero).

Also note that for a critical environment, the agent and target positions used in the proof of Theorem 4.2 give a completion time of $\Omega(n)$ for every monotonic algorithm. However, if a centralized solver were used to assign agents to targets from the same initial positions, then the motion time would be $O(1)$. Hence the distributed solutions given by a monotonic algorithm may severely under-perform when compared to solutions given by the optimal centralized solver. •

B. *The* RENDEZVOUS STRATEGY *and its Drawbacks*

In this section we discuss another approach to solving the target assignment problem that we call the RENDEZVOUS STRATEGY. The strategy, which works only under the full target knowledge assumption, can be described as follows.

```
RENDEZVOUS STRATEGY (for agent $i$)
1 Compute a common meeting point, such as the centroid of the target positions.
2 Move to the meeting point and wait for all other agents to arrive.
3 Once all agents have arrived, broadcast UID$^{[i]}$ and $\mathbf{p}^{[i]}$, and receive UID$^{[k]}$ and $\mathbf{p}^{[k]}$ from all other agents.
4 Compute a complete assignment of target-agent pairs using the MAXIMAL MATCH algorithm and move to
  your assigned target.
```

Since every agent knows the position of all targets, the agents can compute a common meeting point. The time for an agent to reach any meeting point is bounded by $\sqrt{2}|\mathcal{E}(n)|/v_{\text{max}}$, and thus each agent can determine when all other agents have arrived at the meeting point. Once all agents reach the meeting point the communication graph is complete and each agent can broadcast its UID and position to all other agents in one communication round. Then, each agent can use MAXIMAL MATCH to solve a centralized assignment, and all agents end up with the same complete assignment. In addition, since the agents are co-located, this assignment is optimal. Each agent then moves to the target to which it has been assigned. Essentially, this approach turns the distributed problem into a centralized one.

Theorem 4.4 (Time complexity for RENDEZVOUS STRATEGY*):* Consider n agents and n targets in the environment $\mathcal{E}(n)$. In the worst-case, the RENDEZVOUS STRATEGY solves the target assignment problem in $\Theta(\sqrt{|\mathcal{E}(n)|})$ time. Moreover, if the targets and agents are uniformly randomly distributed in $\mathcal{E}(n)$, the completion time is in $\Theta(\sqrt{|\mathcal{E}(n)|})$.

Proof: Since all information can be exchanged in one round, and we are not considering computation time, the completion time is given by the time to reach the meeting point plus the time to go from the meeting point to the assigned target.

To see the worst-case, place all targets at one side of the environment, and all agents at the other side. Then each agent must travel a distance $\Theta(\sqrt{|\mathcal{E}(n)|})$. The distance from the meeting point back to any assigned target is also bounded by $O(\sqrt{|\mathcal{E}(n)|})$. Thus, the worst-case completion time is $\Theta(\sqrt{|\mathcal{E}(n)|})$.

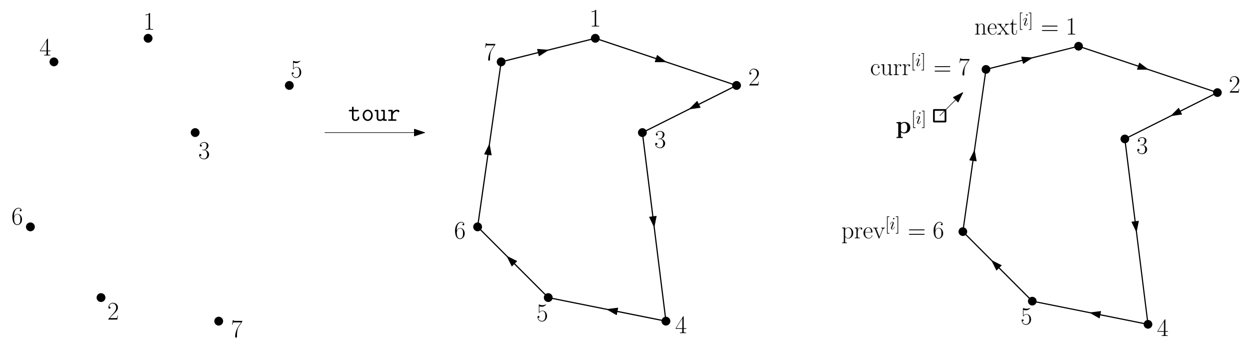
If we uniformly randomly distribute n agents in $\mathcal{E}(n)$, then it is a well known fact (see, for example [21]) that w.h.p., the maximum distance between agents, $\max_{i,j\in\mathcal{I}} \|\mathbf{p}^{[i]} - \mathbf{p}^{[j]}\|$, is in $\Theta(\sqrt{|\mathcal{E}(n)|})$. Thus, one agent must travel a distance of at least $\frac{1}{2}\max_{i,j\in\mathcal{I}} \|\mathbf{p}^{[i]} - \mathbf{p}^{[j]}\| \in \Theta(\sqrt{|\mathcal{E}(n)|})$. Hence, w.h.p., the completion time is in $\Theta(\sqrt{|\mathcal{E}(n)|})$. ■

Remark 4.5 (Drawbacks of RENDEZVOUS STRATEGY*):* From Theorem 4.4 we see that the RENDEZVOUS STRATEGY has better worst-case performance than any monotonic algorithm. Thus, there may be applications in which this is the best algorithm for solving the target assignment problem. However, there are several drawbacks to the algorithm. First, this approach is not a distributed solution in the sense that it requires each agent to acquire information about all other agents in the group, and to solve a centralized assignment problem. Second, the process of meeting to exchange information creates a single point of failure for the system. Third, if we consider an initial configuration where m targets are occupied, then in the RENDEZVOUS STRATEGY all of these targets become unoccupied as the agents travel to the meeting point. Thus, this is not a monotonic algorithm. In fact, if every target is occupied and we run the RENDEZVOUS STRATEGY, all agents leave their targets, move to the meeting point, compute a complete assignment, and move to a new target. This is obviously not the desired behavior in this instance. Fourth, the RENDEZVOUS STRATEGY is ill-suited for heterogeneous situations where agents have widely distinct speeds, or become active at different instants of time; in these situations the RENDEZVOUS STRATEGY essentially reduces the performance of every agent to that of the slowest agent. Fifth, the RENDEZVOUS STRATEGY does not work under the local target sensing assumption, whereas we will provide an algorithm later that does. Finally, in settings where more agent are available than targets, there is hope to complete the target assignment problem in time that is independent of n. The RENDEZVOUS STRATEGY never achieves this time complexity, whereas we will prove this property for one of our proposed algorithms below. •

Because of the drawbacks mentioned in the previous remark, in the remainder of this paper, we look at distributed monotonic algorithms and their performance in solving the target assignment problem.

V. A CONSTANT FACTOR MONOTONIC ALGORITHM IN SPARSE ENVIRONMENTS

We begin by introducing a monotonic algorithm, called the ETSP ASSGMT algorithm, for solving the target assignment problem. This algorithm operates only under the full target knowledge assumption. In this algorithm, each agent precomputes an optimal tour through the n targets, turning the cloud of target points into an ordered ring. Agents then move along the ring, looking for the next available target. When agents communicate, they exchange information on the next available target along the ring. We show that in sparse or critical environments, the ETSP ASSGMT algorithm is within a constant factor of the optimal monotonic algorithm for worst-case initial conditions.



(a) The map tour, creating an ETSP tour of seven targets. (b) The initialization for agent i.

Fig. 2. The initialization process for the ETSP ASSGMT algorithm.

A. *The* ETSP ASSGMT *Algorithm*

The ETSP ASSGMT algorithm is designed under the full target knowledge assumption. In the following description it will be convenient to assume that the target positions are stored in each agents memory as an array, rather than as an unordered set. That is, we replace the target set $\mathcal{Q}$ with the target n-tuple $\mathbf{q} := (\mathbf{q}_1, \dots, \mathbf{q}_n)$, and the local target set $\mathcal{Q}^{[i]}$ with the n-tuple $\mathbf{q}^{[i]} := \mathbf{q}$.[4] The algorithm can be described as follows. For each $i \in \mathcal{I}$, agent i computes a constant factor approximation of the optimal ETSP tour of the n targets in $\mathbf{q}^{[i]}$ (as discussed in Section II-B), denoted $\text{tour}(\mathbf{q}^{[i]})$. We can think of tour as a permutation that reorders the entries of $\mathbf{q}^{[i]}$. This permutation is independent of i since all agents use the same method. An example is shown in Fig. 2(a).

Agent i then replaces its n-tuple $\mathbf{q}^{[i]}$ with $\text{tour}(\mathbf{q}^{[i]})$. Next, agent i computes the index of the closest target in $\mathbf{q}^{[i]}$, and calls it $\text{curr}^{[i]}$. Agent i also maintains the index of the next target in the tour that may be available, $\text{next}^{[i]}$, and first target in the tour before $\text{curr}^{[i]}$ that may be available, $\text{prev}^{[i]}$. Thus, $\text{next}^{[i]}$ is initialized to $\text{curr}^{[i]} + 1 \pmod n$ and $\text{prev}^{[i]}$ to $\text{curr}^{[i]} - 1 \pmod n$. This is depicted in Fig. 2(b). Agent i also maintains the n-tuple, $\text{status}^{[i]}$, which records whether a target is occupied by (assigned to) another agent or not. Letting $\text{status}^{[i]}(j)$ denote the jth entry in the n-tuple, the entries are given by

$$\text{status}^{[i]}(j) = \begin{cases} 0, & \text{if agent } i \text{ knows } \mathbf{q}_j^{[i]} \text{ is assigned to another agent,} \\ 1, & \text{otherwise.} \end{cases} \tag{2}$$

Thus, $\text{status}^{[i]}$ is initialized as the n-tuple $(1, \dots, 1)$. The initialization is summarized in Algorithm 1 of Appendix A.

Agent i then moves toward the target $\text{curr}^{[i]}$ at constant speed $v_{\max} > 0$:

$$\dot{\mathbf{p}}^{[i]} = \begin{cases} v_{\max} \frac{\mathbf{q}^{[i]}_{\text{curr}^{[i]}} - \mathbf{p}^{[i]}}{\|\mathbf{q}^{[i]}_{\text{curr}^{[i]}} - \mathbf{p}^{[i]}\|}, & \text{if } \mathbf{q}^{[i]}_{\text{curr}^{[i]}} \neq \mathbf{p}^{[i]}, \\ 0, & \text{otherwise,} \end{cases} \tag{3}$$

[4]It is possible that the order of the targets in the local sets $\mathbf{q}^{[i]}$ may initially be different. However, given a set of distinct points in $\mathbb{R}^2$, it is always possible to create a unique ordering.

Finally, at each communication round agent i executes the algorithm COMM-RD displayed in Algorithm 2 of Appendix A. The COMM-RD algorithm operates as follows: Agent i, which is heading toward target curr$^{[i]}$, communicates with its neighbors to determine if any other agents are heading toward curr$^{[i]}$. If another agent is heading to curr$^{[i]}$, then the agent closer to curr$^{[i]}$ continues moving toward the target, while the farther agent selects a new target along the tour (ties are broken using UID's). The agents also exchange information on targets that are occupied using the prev, and next variables. The following is a more formal description that omits a few minor technicalities.

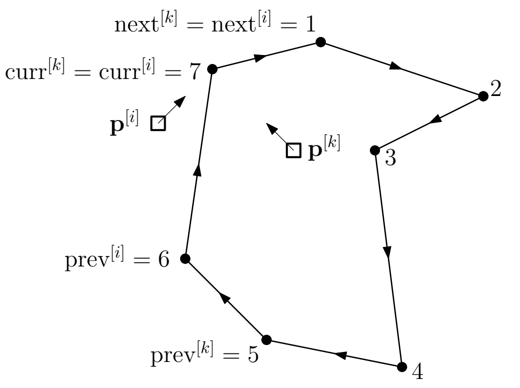
```
Description of COMM-RD for agent $i$
1 Broadcast msg$^{[i]}$, consisting of UID$^{[i]}$, the target indices prev$^{[i]}$, curr$^{[i]}$, and next$^{[i]}$, and the distance to the
  current target, dist$^{[i]}$.
2 for message, msg$^{[k]}$, received do
3     Set status$^{[i]}(j)$ to assigned ('0') for each target $j$ from prev$^{[k]} + 1 \pmod n$ to next$^{[k]} - 1 \pmod n$ not
      equal to curr$^{[i]}$.
4     if curr$^{[i]} =$ curr$^{[k]}$ and dist$^{[i]} >$ dist$^{[k]}$ then Set the status of curr$^{[i]}$ to assigned ('0').
5     if curr$^{[i]} =$ curr$^{[k]}$ and dist$^{[i]} <$ dist$^{[k]}$ then Leave curr$^{[i]}$ unchanged. However, agent $k$ will set curr$^{[k]}$ to a
      new target. This target will be at least as far along the tour as the farther of next$^{[i]}$ and next$^{[k]}$. So, set the
      status of next$^{[i]}$ and next$^{[k]}$ to assigned ('0').
6 Update curr$^{[i]}$ to the next target in the tour with status available ('1'), next$^{[i]}$ to the next available target in the
  tour after curr$^{[i]}$, and prev$^{[i]}$ to the first available target in the tour before curr$^{[i]}$.
```

Fig. 3 gives an example of COMM-RD resolving a conflict between agents i and k, over curr$^{[i]} =$ curr$^{[k]}$. In this figure, all other agents are omitted. In summary, the ETSP ASSGMT algorithm is the triplet consisting of the initialization of each agent (see Algorithm 1), the motion law in Eq. (3), and COMM-RD (see Algorithm 2), which is executed at each communication round.

B. *Correctness and Time Complexity of the* ETSP ASSGMT *Algorithm*

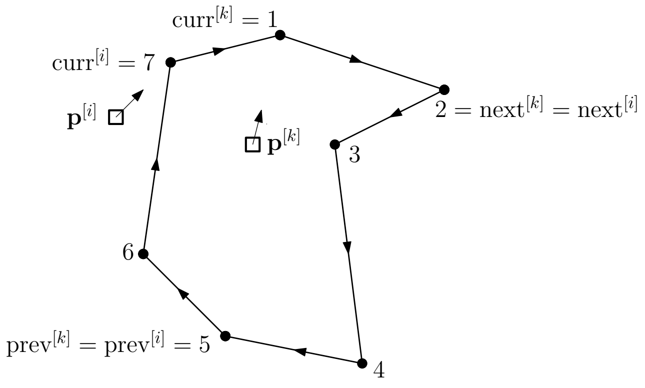
We now present our main result on the ETSP ASSGMT algorithm. Section V-C contains its proof. Recall that the ETSP ASSGMT algorithm requires the full target knowledge assumption.

Theorem 5.1 (Correctness and worst-case bound for ETSP ASSGMT*):* For any initial positions of n agents and n targets in $\mathcal{E}(n)$, ETSP ASSGMT solves the target assignment problem in $O(\sqrt{n|\mathcal{E}(n)|})$ time. In addition, if $\mathcal{E}(n)$ is sparse or critical, then ETSP ASSGMT is within a constant factor of the optimal monotonic algorithm for worst-case initial positions.



$\text{next}^{[k]} = \text{next}^{[i]} = 1$
$\text{curr}^{[k]} = \text{curr}^{[i]} = 7$
$\mathbf{p}^{[i]}$
$\mathbf{p}^{[k]}$
2
3
$\text{prev}^{[i]} = 6$
$\text{prev}^{[k]} = 5$
4

(a) Setup before the conflict over target 7.



$\text{curr}^{[k]} = 1$
$\text{curr}^{[i]} = 7$
$\mathbf{p}^{[i]}$
$\mathbf{p}^{[k]}$
$2 = \text{next}^{[k]} = \text{next}^{[i]}$
3
6
$\text{prev}^{[k]} = \text{prev}^{[i]} = 5$
4

(b) Setup after resolution of the conflict.

Fig. 3. The resolution of a conflict between agents i and k over target 7. Since agent k is closer to target 7 than agent i, agent k wins the conflict.

C. *Proof s for Statements about the* ETSP ASSGMT *Algorithm*

To prove Theorem 5.1 we introduce a few definitions. We say that agent $i \in \mathcal{I}$ is *assigned* to target $\mathbf{q}_j^{[i]}$, $j \in \mathcal{I}$, when $\text{curr}^{[i]} = j$. In this case, we also say target j is assigned to agent i. We say that agent $i \in \mathcal{I}$ *enters a conflict* over the target $\text{curr}^{[i]}$, when agent i receives a message, $\text{msg}^{[k]}$, with $\text{curr}^{[i]} = \text{curr}^{[k]}$. Agent i *loses the conflict* if agent i is farther from $\text{curr}^{[i]}$ than agent k, and *wins the conflict* if agent i is closer to $\text{curr}^{[i]}$ than agent k, where ties are broken by comparing UIDs.

The following lemma is a direct result of the facts that the environment is bounded for each $n \in \mathbb{N}$, and that the agents move at constant speed $v_{\text{max}} > 0$.

Lemma 5.2 (Conflict in finite time): Consider any communication range $r_{\text{comm}} > 0$, and any fixed number of agents $n \in \mathbb{N}$. If, for two agents i and k, $\text{curr}^{[i]} = \text{curr}^{[k]}$ at some time $t_1 \geq 0$, then agent i (and likewise, agent k) will enter a conflict over $\text{curr}^{[i]}$ in finite time.

In order to prove correctness, we require a few properties of the ETSP ASSGMT algorithm.

*Lemma 5.3 (*ETSP ASSGMT *properties):* During an execution of the ETSP ASSGMT algorithm, the following statements hold for agent $i \in \mathcal{I}$:

(i) the current target $\text{curr}^{[i]}$ satisfies $\text{status}^{[i]}(\text{curr}^{[i]}) = 1$;

(ii) $\text{status}^{[i]}(j) = 0$ for each $j \in \{\text{prev}^{[i]} + 1, \text{prev}^{[i]} + 2, \dots, \text{next}^{[i]} - 1\} \setminus \{\text{curr}^{[i]}\} \pmod n$;

(iii) $\text{status}^{[i]}(j) = 0$ only if target j is assigned to some agent $k \neq i$;

(iv) if $\text{status}^{[i]}(j) = 0$ at some time t_1, then $\text{status}^{[i]}(j) = 0$ for all $t \geq t_1$;

(v) if agent i receives $\text{msg}^{[k]}$ during a communication round, then agent i will set $\text{status}^{[i]}(j) = 0$ for each $j \in \{\text{prev}^{[k]} + 1, \dots, \text{next}^{[k]} - 1\} \setminus \{\text{curr}^{[i]}\} \pmod n$.

Proof: Statements (i) and (iv) and (v) follow directly from the initialization and COMM-RD.

Statement (ii) is initially satisfied since $\text{prev}^{[i]} + 1 = \text{curr}^{[i]} = \text{next}^{[i]} - 1$ implies that $\{\text{prev}^{[i]} + 1, \dots, \text{next}^{[i]} - 1\} \setminus$

$\{\text{curr}^{[i]}\} = \emptyset$. Assume that statement (ii) is satisfied before the execution of COMM-RD. At the end of COMM-RD, $\text{prev}^{[i]}$ is updated to the first target before $\text{curr}^{[i]}$ in the tour with status available ('1'). If $\text{status}^{[i]}(\text{curr}^{[i]}) = 1$, then $\text{curr}^{[i]}$ remains unchanged. If $\text{status}^{[i]}(\text{curr}^{[i]}) = 0$, then $\text{curr}^{[i]}$ is increased to the first target with status available ('1'). Finally, $\text{next}^{[i]}$ is set to the first target after $\text{curr}^{[i]}$ that is available. Thus, at the end of COMM-RD the status of $\text{prev}^{[i]}$, $\text{curr}^{[i]}$ and $\text{next}^{[i]}$ are available, and $\text{status}^{[i]}(j) = 0$ for each target $j \in \{\text{prev}^{[i]} + 1, \dots, \text{next}^{[i]} - 1\} \setminus \{\text{curr}^{[i]}\} \pmod n$.

Statement (iii) is also initially satisfied since $\text{status}^{[i]} = \mathbf{1}_n$ for each $i \in \mathcal{I}$. Assume Statement (iii) is satisfied before the execution of COMM-RD and that during this communication round agent i changes the status of a target j to assigned ('0'). We show that Statement (iii) is still satisfied upon completion of the execution of COMM-RD. In order for $\text{status}^{[i]}(j)$ to be changed, agent i must have received a message, $\text{msg}^{[k]}$, for which one of the following cases is satisfied: (1) Target $j \neq \text{curr}^{[i]}$ lies between $\text{prev}^{[k]}$ and $\text{next}^{[k]}$ on the tour; (2) There is a conflict between agents i and k over target j that agent i loses; or, (3) There is a conflict between agents i and k that agent i wins and $\text{next}^{[i]} = j$ or $\text{next}^{[k]} = j$.

In Case (1) either $\text{status}^{[k]}(j) = 0$ or $\text{curr}^{[k]} = j$, and thus target j is assigned. In Case (2) agent k won the conflict implying $\text{curr}^{[k]} = j$ entering the communication round. Thus after the communication round, $\text{curr}^{[i]} \neq j$ and target j is assigned to another agent. In Case (3), $\text{curr}^{[i]} = \text{curr}^{[k]} \neq j$, and agent k loses the conflict. In this case, agent k will change $\text{curr}^{[k]}$ to the next available target on its tour. All targets from $\text{prev}^{[k]} + 1$ to $\text{next}^{[k]} - 1$ have been assigned. Also, during the communication round, agent k will receive $\text{msg}^{[i]}$ and determine that all targets from $\text{prev}^{[i]} + 1$ to $\text{next}^{[i]} - 1$ are assigned. Thus, the next available target is at least as far along the tour as the farther of $\text{next}^{[i]}$ and $\text{next}^{[k]}$. Thus, after the communication round, both $\text{next}^{[i]}$ and $\text{next}^{[k]}$ are assigned. ■

With these properties we are now ready to prove Theorem 5.1.

Theorem 5.1: We begin by proving the correctness of the ETSP ASSGMT algorithm. Assume by way of contradiction that at some time $t_1 \geq 0$ there are $J \in \{1, \dots, n-1\}$ targets unassigned, and for all time $t \geq t_1$, J targets remain unassigned. Since the algorithm is monotonic, the same $n - J$ assigned targets remain assigned for all time, and thus it must be the same J targets that remain unassigned for all $t \geq t_1$. Let $\mathcal{J}$ denote the index set of the J unassigned targets. From our assumption, and by Lemma 5.3 (iii), for every $t \geq t_1$ and for every $i \in \mathcal{I}$, $\text{status}^{[i]}(j) = 1$ for each $j \in \mathcal{J}$. Now, among the $n - J$ assigned targets there is at least one target to which two or more agents are assigned. Consider one such target, call it j_1, and consider an agent i_1 with $\text{curr}^{[i_1]} = j_1$. By Lemma 5.2, agent i_1 will enter a conflict over j_1 in finite time. Let us follow the loser of this conflict. The losing agent, call it i_2, will set $\text{status}^{[i_2]}(j_1) = 0$ and will move to the next target in the tour it believes may be available, call it j_2. Now, we know j_2 is not in $\mathcal{J}$, for if it were $J - 1$ targets would be unassigned contradicting our assumption. Moreover, by Lemma 5.3 (i), $j_2 \neq j_1$. Thus, agent i_2 will enter a conflict over j_2 in finite time. After this conflict the losing agent, call it i_3, will set $\text{status}^{[i_3]}(j_2) = 0$ (because it lost the conflict), and from Lemma 5.3 (v), $\text{status}^{[i_3]}(j_1) = 0$. Again, agent i_3's next target j_3 must not be in $\mathcal{J}$, for if it were we would have a contradiction. Thus, repeating this argument $n - J$ times we have that agent i_{n-J} loses a conflict over j_{n-J}. After this conflict, we have $\text{status}^{[i_{n-J}]}(j_k) = 0$ for each $k \in \{1, \dots, n - J\}$, where $j_{k_1} = j_{k_2}$ if and only if $k_1 = k_2$.

In other words, agent i_{n-J} knows that all $n-J$ assigned targets have indeed been assigned. Also, by our initial assumption, $\text{status}^{[i_{n-J}]}(j) = 1$ for each $j \in \mathcal{J}$. By Lemma 5.3 (i), agent i_{n-J}'s new current target must have status available ('1'). Therefore, it must be that agent i_{n-J} will set $\text{curr}^{[i_{n-J}]}$ to a target in $\mathcal{J}$. Thus, after a finite amount of time $J-1$ targets are unassigned, a contradiction.

We now prove the upper bound on the performance of the ETSP ASSGMT algorithm. First notice the following: Consider the optimal ETSP tour through all n targets. This provides an ordering in which the n targets are visited. Now, suppose k targets are removed from the tour, and the $n-k$ remaining targets are visited in the order they appeared in the n-target tour. In general, this is not the optimal tour through the $n-k$ points. However, by the triangle inequality, the length of the tour is no longer than that of the tour through all n points. Because of this, in the worst-case some agent must travel to its nearest target, and then around its entire ETSP tour, losing a conflict at each of the first $n-1$ targets in the tour. For any initial agent and target positions, the distance to the nearest target is $O(\sqrt{|\mathcal{E}(n)|})$. Since the length of each agent's tour is a constant factor approximation of the optimal, the tour length is $O(\sqrt{n\mathcal{E}(n)})$ (see Theorem 2.1). The agent will not follow the ETSP tour exactly because it may enter conflicts before actually reaching the targets; however, by the triangle inequality, the resulting path cannot be longer than the ETSP tour. Hence, the total distance traveled is in $O(\sqrt{n\mathcal{E}(n)})$, and since the agents move at constant speed, the completion time is in $O(\sqrt{n\mathcal{E}(n)})$. Combining this with Theorem 4.2 we see that in critical or sparse environments the completion time is in $\Theta(\sqrt{n\mathcal{E}(n)})$.

■

VI. AN CONSTANT FACTOR MONOTONIC ALGORITHM IN DENSE ENVIRONMENTS

In the previous section we presented the ETSP ASSGMT algorithm which operates only with full target knowledge but has provably good performance in sparse and critical environments. In this section we introduce a monotonic algorithm called the GRID ASSGMT algorithm which operates under both full target knowledge and local target sensing with $r_{\text{sense}} \geq \sqrt{2/5}r_{\text{comm}}$. In this algorithm, the agents partition the environment into cells. Agents then determine local maximum assignments, and elect a leader in the cell which they occupy. Through communication between leaders of adjacent cells, each leader obtains estimates of the location of free targets, and uses this information to guide unassigned agents to free targets. We show that in critical or dense environments, the GRID ASSGMT algorithm is within a constant factor of the optimal monotonic algorithm for worst-case initial conditions. In addition, we characterize the stochastic performance of the GRID ASSGMT algorithm.

A. *The* GRID ASSGMT *Algorithm*

In the GRID ASSGMT algorithm we make either the full target knowledge assumption (i.e., $\mathcal{Q}^{[i]} := \mathcal{Q}$), or the local target sensing assumption with $r_{\text{sense}} \geq \sqrt{2/5}r_{\text{comm}}$. In addition we assume each agent knows the environment $\mathcal{E}(n)$. Each agent partitions the environment into b^2 equally sized square cells, where $b \in \mathbb{N}$. It then labels the cells like entries in a matrix, so cell $C(r,c)$ resides in the rth row and cth column, as shown in Fig. 4. Since the agents started with the same information, they all create the same partition. The quantity b is chosen so that an agent in

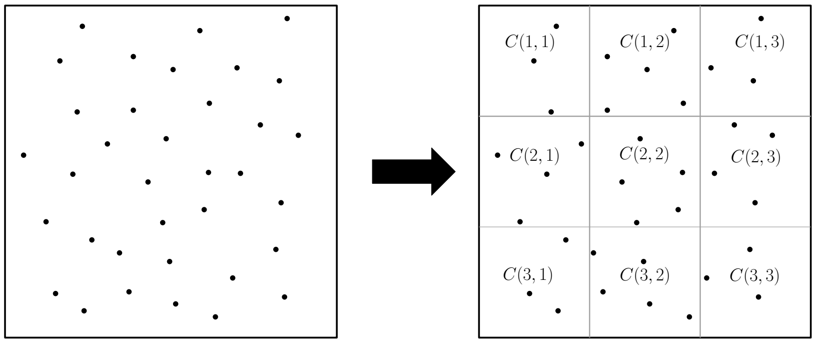


Fig. 4. Partitioning the environment $\mathcal{E}(n)$, containing 35 targets, into $b^2 = 9$ cells.

cell $C(r,c)$ is within communication range of any agent in cells $C(r,c)$, $C(r-1,c)$, $C(r+1,c)$, $C(r,c-1)$, and $C(r,c+1)$. In light of Lemma 2.3, we see that this is satisfied when $b = \lceil \sqrt{5|\mathcal{E}(n)|}/r_{\text{comm}} \rceil$. Note that with $r_{\text{sense}} \geq \sqrt{2/5} r_{\text{comm}}$ an agent in cell $C(r,c)$ can sense the position of all targets in that cell. We now outline the GRID ASSGMT algorithm.

Outline of the GRID ASSGMT algorithm

***Initialization and role assignment*:** Each agent partitions the environment as described above. In each cell, agents find a maximum assignment between agents and targets occupying the cell, and assigned agents elect a leader among them. Accordingly, agents are labeled leader, unassigned, or assigned non-leader. According to their role, agents allocate certain variables describing their location and their knowledge about target assignments.

Assigned non-leader agents: Each assigned non-leader agent move to its assigned target and goes silent.

Cell leaders: Each cell leader estimates the number of available targets in all cells below it in its column. The leader i of cell $C(r,c)$ stores this estimate in the variable $\Delta^{[i]}_{\text{blw}}(r,c)$; to maintain the estimates, cell leaders communicate to the cell leader in the cell directly above it. Additionally, each cell leader in the top row communicates to the cell leader in the cell directly to the right, to obtain an estimate of the number of available targets in all columns to the right (denoted $\Delta^{[j]}_{\text{rght}}(1,c)$ for leader j of cell $C(1,c)$).

Unassigned agents: Each unassigned agent seeks a free target by entering cells and querying their respective leaders. The motion of unassigned agents is illustrated in Fig. 5. Assuming no communication with the leaders, the nominal order in which an unassigned agent visits all cells of the grid is shown in the left-hand figure. The way in which this path is shortened as the unassigned agent receives available target estimates from cell leaders is shown on the right-hand figure.

Remark 6.1 (Computations performed by cell leaders): If agent i is the leader of cell $C(r,c)$, it computes $\Delta^{[i]}(r,c)$, which is (# of targets) – (# of agents) in $C(r,c)$. In addition, leader i maintains $\Delta^{[i]}_{\text{blw}}(r,c)$, which is an estimate of (# of targets) – (# of agents) in cells $C(r+1,c)$ to $C(b,c)$. This quantity must be estimated because agent i

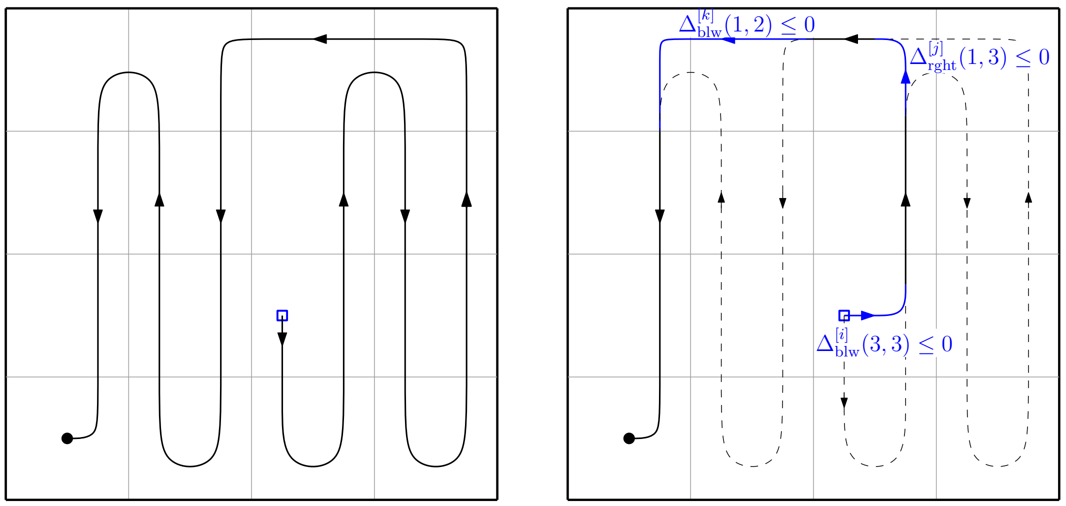

$\Delta^{[k]}_{\text{blw}}(1,2) \leq 0$
$\Delta^{[j]}_{\text{rght}}(1,3) \leq 0$
$\Delta^{[i]}_{\text{blw}}(3,3) \leq 0$

Fig. 5. The figure on the left shows the nominal order in which an agent, depicted as a blue square, searches the cells in the absence of communication. The blue lines on the right-hand figure show how this path is shortened by the non-positive estimates from leader i of $C(3,3)$, leader j of $C(1,3)$ and leader k of $C(1,2)$.

does not initially know the number of agents in cells $C(r+1,c)$ to $C(b,c)$. The variable $\Delta^{[i]}_{\text{blw}}(r,c)$ is initialized to $+\infty$ (i.e., a very large positive number) for the leaders in rows 1 to $b-1$, and to 0 for the leaders in row b. Then, at each communication round agent i updates its estimate by communicating with the leaders in cells $C(r-1,c)$ and $C(r+1,c)$:

```
1 Send msg$^{[i]} := \Delta^{[i]}_{\text{blw}}(r,c) + \Delta^{[i]}(r,c)$ to leader in cell $C(r-1,c)$ and receive msg$^{[k]}$ from
  agent $k$, the leader of $C(r+1,c)$.
2 Set $\Delta^{[i]}_{\text{blw}}(r,c) :=$ msg$^{[k]} = \Delta^{[k]}_{\text{blw}}(r+1,c) + \Delta^{[k]}(r+1,c)$.
```

This update procedure is depicted in Fig. 6. A leader j of cell $C(1,c)$ in the top row uses a similar method to maintain the estimate $\Delta^{[j]}_{\text{rght}}(1,c)$. It should be noted that as unassigned agents enter and exit cells, the actual values of Δ_{blw} and Δ_{rght} change. Thus, there is a procedure whereby agents send `enter` and `exit` messages to cell leaders, so that they can maintain their estimates. This is detailed in the algorithms of Appendix B. •

Remark 6.2 (Motion performed by unassigned agents): Let us describe the unassigned agents motion in more detail. First, each unassigned agent seeks a free target in its column as follows. It queries the leader of its current cell about free targets in its column, below its current cell. If the leaders estimate $\Delta^{[i]}_{\text{blw}}(r,c)$ is positive, then the agent moves down the column. Otherwise, the agent moves up the column. While moving down, upon entering a new cell the agent first queries the cell leader on free targets in the cell, and then on free targets in cells below. If the agent starts moving up the column, then it only queries cell leaders on free targets in the cell (since it knows no targets are free in the cells below).

Second, if the agent reaches the top cell of its column, then the column contains no free targets. To transfer to a new column, the agent queries the leader of the top cell about free targets in all columns to the right. If the

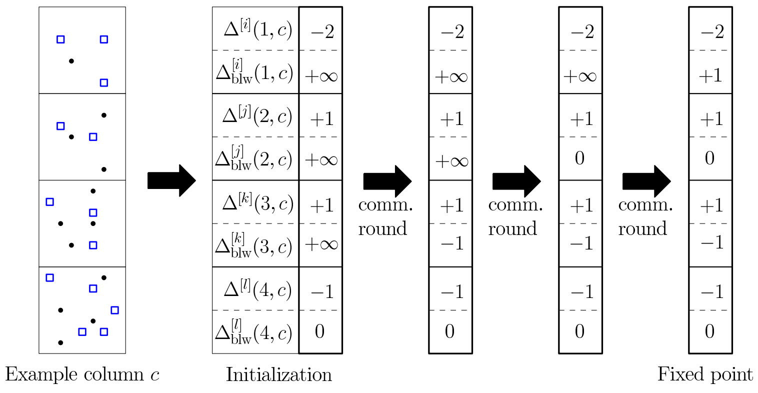


Fig. 6. In the example column c, blue squares depict agents, and black disks depict targets. The figure shows how the estimates Δ and Δ_{blw} are initialized and updated by leaders i of $C(1,c)$, j of $C(2,c)$, k of $C(3,c)$, and l of $C(4,c)$. After three communication rounds the estimates have converged to the true values.

leaders estimate $\Delta^{[j]}_{\text{blw}}(1,c)$ is positive, then the agent moves to the right; otherwise, the agent moves to the left. Upon reaching the cell to the left or right, the agent recommences the column procedure. •

In summary, a detailed description of the components of the GRID ASSGMT algorithm is given in Appendix B: all variables maintained by the agents are listed in Table I; the initialization and role assignment is performed by the ROLE ASSGMT algorithm, see Algorithm 3; the behavior of the cell leaders and of the unassigned agents are described by the LEADER and by the UNASSIGNED algorithm, see Algorithms 4 and 5, respectively.

Remark 6.3 (Using a single transfer row): In our description of the GRID ASSGMT algorithm, agents use the top row to transfer to a new column. This choice of "transfer row" is arbitrary and the top row was chosen for simplicity of presentation. Intuitively, it seems the middle row is a more efficient choice. The upcoming analysis shows that such a choice does not affect the algorithm's asymptotic performance. The reason we require unassigned agents to use a single transfer row is because it allows for cell leaders to easily maintain up-to-date estimates of unassigned agent and free target locations. To understand this, suppose that there were two transfer rows, row 1 and row b, and that two unassigned agents simultaneously transfer from column $c-1$ to column c, one using row 1, and the other using row b. Then, it would take $b \in \Theta(\sqrt{|\mathcal{E}(n)|})$ communication rounds for the leader in cell $C(1,c)$ to become aware that an unassigned agent transferred using row b, implying that leader estimates are not up-to-date. To overcome this, one would need to halt unassigned agent motion until leader estimates have been updated; a process which would require more leader communication. In addition, using more transfer rows does not appear to change the asymptotic performance (although the constant factor could be significantly reduced since the algorithm would rely more heavily on communication than agent motion). Thus, we have utilized a single transfer row to minimize excess communication, and avoid introducing more complexity in the algorithm. In addition this helps to avoid wireless congestion issues, which can become very significant as the number of agents becomes large [22].

•

Remark 6.4 (Details of the GRID ASSGMT *algorithm):* (1) Agents move at speed v_{max}, and to transfer between cells agents move toward the center of the new cell. (2) If an agent or target lies on the boundary between cells, a simple tie breaking scheme is used assign it to a cell. (3) In our presentation, we implicitly assumed that every cell initially contains at least one agent and one target. If a cell has no targets, then any agents initially in the cell leave, and the empty cell is then ignored. If a cell initially contains targets but no agents, then the first agents to enter the cell run the MAXIMAL MATCH algorithm and a leader is elected. •

B. Correctness and Time Complexity of the GRID ASSGMT *Algorithm*

We now present our main results on the GRID ASSGMT algorithm. Section VI-C contains their proofs. Recall that the GRID ASSGMT algorithm operates under the full target knowledge assumption, and the local target sensing assumption with $r_{\text{sense}} \geq \sqrt{2/5}r_{\text{comm}}$.

Theorem 6.5 (Correctness and worst-case bound for GRID ASSGMT*):* For any initial positions of n agents and n targets in $\mathcal{E}(n)$, the GRID ASSGMT algorithm solves the target assignment problem in $O(|\mathcal{E}(n)|)$ time. In addition, if $\mathcal{E}(n)$ is dense or critical, then the GRID ASSGMT algorithm is within a constant factor of the optimal monotonic algorithm for worst-case initial conditions.

*Remark 6.6 (*GRID ASSGMT *algorithm vs.* ETSP ASSGMT *algorithm):* The worst-case bound for the ETSP ASSGMT algorithm in Theorem 5.1 was $O(\sqrt{|\mathcal{E}(n)|n})$. Thus, in sparse environments the ETSP ASSGMT algorithm performs better, where as in dense environments the GRID ASSGMT algorithm performs better. In critical environments, the bounds are equal. Thus, the two algorithms are complementary. In practice, a robot can determine which algorithm to run by comparing the area of the environment $|\mathcal{E}(n)|$ to the area of n disks of radius r_{comm}. That is, given n, $\mathcal{E}(n)$ and r_{comm}, a robot could use a rule such as the following: if $|\mathcal{E}(n)| > \pi r_{\text{comm}}^2 n$, then execute the ETSP ASSGMT algorithm, else if $|\mathcal{E}(n)| < \pi r_{\text{comm}}^2 n$, then execute the GRID ASSGMT algorithm. •

In the following theorem we will see that for randomly placed targets and agents, the performance of the GRID ASSGMT algorithm is considerably better than in the worst-case.

Theorem 6.7 (Stochastic time complexity): Consider n agents and n targets, uniformly randomly distributed in $\mathcal{E}(n)$. Then the GRID ASSGMT algorithm solves the target assignment problem in $O(\sqrt{|\mathcal{E}(n)|})$ time with high probability if

$$|\mathcal{E}(n)| \leq \frac{r_{\text{comm}}^2}{5} \frac{n}{\log n + \gamma(n)},$$

where γ is any function such that $\gamma(n) \to +\infty$ as $n \to +\infty$.

Remark 6.8 (Generalization of Theorem 6.7): The bound in Theorem 6.7 holds not only for uniformly randomly generated initial positions, but for any initial positions such that every cell contains at least one target and at least one agent.

Theorem 6.9 (Stochastic time complexity, cont'd): Consider n agents and $n/\log n$ targets, uniformly randomly distributed in $\mathcal{E}(n)$. Then the GRID ASSGMT algorithm solves the target assignment problem in $O(1)$ time with high probability if there exists $K > 1/\log(4/\mathrm{e})$, such that

$$|\mathcal{E}(n)| \leq \frac{r_{\text{comm}}^2}{5} \frac{n}{K \log n}.$$

C. *Proofs for Statements about the* GRID ASSGMT *Algorithm*

In this section we prove the results presented in Section VI-B. The leaders of each cell maintain estimates of the difference between the number of targets and agents in various parts of the grid. In order to talk about the convergence of these estimates we introduce a few quantities. Let tar(r, c) denote the number of targets in $C(r, c)$. Let $\Delta(r, c)(t)$ denote the difference between tar(r, c) and the number of agents with currcell$^{[i]} = C(r, c)$ at time $t > 0$. (Notice the lack of superscript on $\Delta(r, c)(t)$, when compared to agent i's estimate of the quantity, $\Delta^{[i]}(r, c)$.) Recall that in our model, communication round k occurs instantaneously at time t_k. Thus, we let t_k^- denote start of the round, and t_k^+, its completion, and so $\Delta^{[i]}(r, c)(t_k^+)$ denotes value of $\Delta^{[i]}(r, c)$ at the completion of communication round k.

Lemma 6.10 (Convergence of estimates): During an execution of the GRID ASSGMT algorithm, if agent $i \in \mathcal{I}$ is the leader of cell $C(r, c)$ then for each communication time t_k, $k \in \mathbb{N}$:

(i) $\Delta^{[i]}(r, c)(t_k^+) = \Delta(r, c)(t_k)$;

(ii) $\Delta_{\text{blw}}^{[i]}(r, c)(t_k^+) \geq \sum_{r^*=r+1}^{b} \Delta(r^*, c)(t_k)$;

(iii) if $k > b$ and each cell in column c contains a leader, then $\Delta_{\text{blw}}^{[i]}(r, c)(t_k^+) = \sum_{r^*=r+1}^{b} \Delta(r^*, c)(t_k)$.

Proof: To see part (i) notice that each agent $j \in \mathcal{I}$ initially sets currcell$^{[j]}$ to the cell it occupies. The leader of cell $C(r, c)$, call it agent i, can communicate with all agents in its cell, and it knows the number of targets in $C(r, c)$. Thus, at t_1 agent i counts the agents in its cell, and correctly calculates $\Delta^{[i]}(r, c)(t_1^+) = \Delta(r, c)(t_1)$. Assume that $\Delta^{[i]}(r, c)$ is correct at t_{k-1}^+. We will show that it is correct at t_k^+. If at t_k^- agent j changes currcell$^{[j]}$ to $C(r, c)$, then it must either be in $C(r-1, c)$ or $C(r+1, c)$, or if $r = 1$, possibly $C(r, c-1)$ and $C(r, c+1)$. Upon changing currcell$^{[j]}$ to $C(r, c)$, agent j sends an `enter` message to the leader of $C(r, c)$, and by Lemma 2.3 the leader will receive it at t_k. Likewise, if an agent changes currcell$^{[j]}$ from $C(r, c)$ to another cell, the agent must be in cell $C(r, c)$. Thus, when this agent sends the `exit` message, the leader of $C(r, c)$ will receive it at t_k. Hence, after the leader updates $\Delta^{[i]}(r, c)$ (Step 8 of LEADER), it will have $\Delta^{[i]}(r, c)(t_k^+) = \Delta(r, c)(t_k)$.

The proof of (ii) is as follows. Notice that we can write the sum $\sum_{r^*=r+1}^{b} \Delta(r^*, c)(t_k)$ as

$$\sum_{r^*=r+1}^{b} (\Delta(r^*, c)(t_{k-1})) + \text{enter}(t_{k-1}, t_k) - \text{exit}(t_{k-1}, t_k). \tag{4}$$

where enter(t_{k-1}, t_k) is the number of agents that entered cells $C(r+1, c), \ldots, C(b, c)$ between time t_{k-1} and time t_k, and exit(t_{k-1}, t_k) is the number that exited.

Let agent i be the leader of cell $C(r, c)$. Agent i initializes $\Delta_{\text{blw}}^{[i]}(r, c)$ to $+\infty$, so the inequality is satisfied initially. Assume (ii) is satisfied at t_{k-1}^+. We will show that it is satisfied at t_k^+. If there is no leader in $C(r+1, c)$, then agent i will not receive a message. In this case one of two updates occurs: 1) If an unassigned agent enters cell $C(r, c)$ from cell $C(r+1, c)$ then agent i sets $\Delta_{\text{blw}}^{[i]}(r, c) := 0$ (Step 10 of LEADER). But, from UNASSIGNED, an agent moves up a column only if there are no available targets below, and thus the inequality is satisfied at t_k^+. Alternatively, 2) agent i leaves $\Delta_{\text{blw}}^{[i]}(r,c)$ unchanged, and thus the inequality will be satisfied at t_k^+.

The other case is that leader j is in cell $C(r+1,c)$, and agent i receives the message

$$\Delta_{\text{blw}}^{[j]}(r+1,c)(t_{k-1}^+) + \Delta^{[j]}(r+1,c)(t_{k-1}^+), \tag{5}$$

But by assumption

$$\Delta_{\text{blw}}^{[j]}(r+1,c)(t_{k-1}^+) \geq \sum_{r^*=r+2}^{b} (\Delta(r^*,c)(t_{k-1}))$$

and from (i), $\Delta^{[j]}(r+1,c)(t_{k-1}^+) = \Delta(r+1,c)(t_{k-1})$. Thus, Eq. (5) is no smaller than $\sum_{r^*=r+1}^{b} \Delta(r^*,c)(t_{k-1})$. But, when agent i receives the message in Eq. (5), it adds $\texttt{enter}(t_{k-1},t_k)$ and subtracts $\texttt{exit}(t_{k-1},t_k)$ (see Step 9 of LEADER). Thus, from Eq. (4), the inequality is satisfied at t_k^+.

In light of the proof for (ii), we see that to prove (iii) we need only show that for all $k \geq b$, the message in Eq. (5) equals $\sum_{r^*=r+1}^{b} \Delta(r^*,c)(t_{k-1})$. We do this by induction. Notice that in cell $C(b,c)$, $\Delta_{\text{blw}}^{[j]}(b+1,c)(t_{k-1}^+) = 0$, and so (iii) holds trivially for $k > 0$. In cell $C(b-1,c)$, for $k > 1$, the message in Eq. (5) becomes $\Delta^{[j]}(b,c)(t_{k-1}^+)$, which by (i) equals $\Delta(b,c)(t_{k-1})$. Thus (iii) holds for cell $C(b-1,c)$ and $C(b,c)$ for all $k > 1$. Assume that (iii) holds for $C(r+1,c),\dots,C(b,c)$ at time t_{k-1}^+, where $k > b-r$. We will show it holds for $C(r,c)$ at time t_k^+. Since (iii) holds for cell $C(r+1,c)$ at t_{k-1}, the first term in Eq. (5) is $\sum_{r^*=r+2}^{b} \Delta(r^*,c)(t_{k-1})$, and from (i), the second term is $\Delta(r+1,c)(t_{k-1})$. Thus, the message is $\sum_{r^*=r+1}^{b} \Delta(r^*,c)(t_{k-1})$. ■

We have an analogous result for the convergence of $\Delta_{\text{rght}}^{[i]}(r,c)$. It follows directly from Lemma 6.10 (i) and (iii) and the fact that $\Delta_{\text{rght}}^{[i]}(c)$ is initially overestimated.

Lemma 6.11 (Convergence of estimates, cont'd): If agent $i \in \mathcal{I}$ is the leader of cell $C(1,c)$, then for each communication time t_k, $k \in \mathbb{N}$,

(i) $\Delta_{\text{rght}}^{[i]}(c)(t_k^+) \geq \sum_{c^*=c+1}^{b} \sum_{r^*=1}^{b} \Delta(r^*,c^*)(t_k)$;

(ii) if each cell contains a leader and if $k > 2b$, $\Delta_{\text{rght}}^{[i]}(c)(t_k^+) = \sum_{c^*=c+1}^{b} \sum_{r^*=1}^{b} \Delta(r^*,c^*)(t_k)$.

With these lemmas we will now prove theorems 6.5, 6.7, and 6.9.

Theorem 6.5: We begin by proving the correctness of the GRID ASSGMT algorithm. Assume by way of contradiction that $J \in \{1,\dots,n-1\}$ targets remain unassigned for all time and thus $B \in \{1,\dots,J\}$ cells contain unassigned targets. By construction of the GRID ASSGMT algorithm, an assigned target never becomes unassigned. Thus, the same targets remain unassigned for all time. Let $\mathcal{C}$ denote the set of cells containing these unassigned targets.

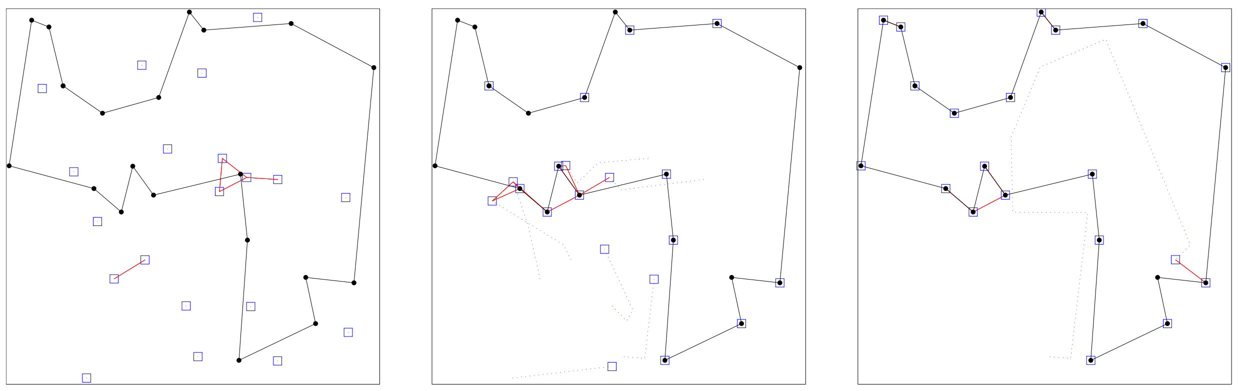
Consider a cell $C(r,c) \in \mathcal{C}$. If $C(r,c)$ does not contain a leader, then it has never been entered by an agent. If it does contain a leader, then taravail$^{[i]}(r,c)$ contains the available targets. Thus if there is an unassigned agent in cell $C(r,c) \in \mathcal{C}$, then upon querying the leader (or if there is no leader, electing a leader), at least one of the targets in $C(r,c)$ will become assigned, contradicting our assumption. Likewise, for each cell $C(r,c) \notin \mathcal{C}$, either there is a leader and taravail$^{[i]}(r,c) = \emptyset$, or there are no targets in the cell.

Now, consider an unassigned agent i, in cell $C(r,c) \notin \mathcal{C}$. Agent i must never enter a cell in $\mathcal{C}$, for if it did an unassigned target would become assigned, a contradiction. We will show this is not possible. According to the UNASSIGNED algorithm, agent i travels down its current column, querying the leader of each cell for available targets in the cell and in cells below. By Lemma 6.10(ii) agent i will only travel back up the column if all targets in cells below have been assigned. After traveling back up the column, if there are no available targets in the top cell in the column, agent i will set $\text{colstatus}^{[i]}(c) = 0$ and will never enter column c again. By Lemmas 6.10 and 6.11, agent i will travel down each column that may possibly have a free target. Thus, at some point agent i necessarily will enter a column containing a cell in $\mathcal{C}$. Hence, either agent i, or another assigned agent will enter the cell in $\mathcal{C}$ at which point the number of assigned targets will increase by at least one, a contradiction.

We now prove the upper bound on the performance of the GRID ASSGMT algorithm. In the worst case, the targets are positioned such that leaders cannot exchange any information about availability of targets. Then, in the worst case an agent, call it i, must visit all b^2 cells before reaching an unassigned target. In the worst case agent i will travel up and down once in every column in the grid, and back and forth once along the top of the grid. In each cell, agent i will query the leader for available targets. If there is no leader in the cell, then agent i will solve a maximum matching among agents that entered at the same time as it, and one of them will become the leader. In either case, the time spent in each cell is $O(1)$. The length of each column is $\sqrt{|\mathcal{E}(n)|}$, and thus the worst-case travel distance is bounded by $2\sqrt{|\mathcal{E}(n)|}(b+2) \in O(|\mathcal{E}(n)|)$. Since the agent moves at constant speed v_{max}, the time for the last agent to reach its final target is in $O(|\mathcal{E}(n)|)$. ■

Theorem 6.7: From Lemma 2.3, $b \leq \lceil \sqrt{n/(\log n + \gamma(n))} \rceil$, where $\gamma(n) \to +\infty$ as $n \to +\infty$. From Theorem 2.2 when we uniformly randomly distribute n targets and n agents into b^2 cells, w.h.p. each cell contains at least one agent, and one target. The maximum matching and leader election in the ROLE ASSGMT algorithm can be performed in $O(1)$ time. Thus in $O(1)$ time there will be a leader in every cell. By Lemma 6.10(iii), in $b \in O(\sqrt{|\mathcal{E}(n)|}$ communication rounds, every leader will know the difference between the number of agents and the number of targets in the cells below it. Thus after $O(\sqrt{|\mathcal{E}(n)|})$ time, the leader of each cell will only let an agent move further down the column if it knows the agent will find an assignment. Also, by Lemma 6.11(ii) after $O(\sqrt{|\mathcal{E}(n)|})$ time, each leader in the top row will only send agents right if there are available targets to the right. Thus, in the worst case, an agent may have to travel out of its own column, across the top column, and then down a new column in order to find its target. This distance is in $O(\sqrt{|\mathcal{E}(n)|})$, and since the agent spends $O(1)$ time in each cell, the time complexity is in $O(\sqrt{|\mathcal{E}(n)|})$. Thus the total time complexity is in $O(\sqrt{|\mathcal{E}(n)|}) + O(\sqrt{|\mathcal{E}(n)|}) \in O(\sqrt{|\mathcal{E}(n)|})$ time. ■

Theorem 6.9: From Lemma 2.3, there are $b^2 \leq \lceil \sqrt{(n/K \log n)} \rceil^2$ cells, where K is a constant satisfying $K > 1/\log(4/\mathrm{e})$. Equivalently, we can write $b^2 = \frac{1}{c(n)} \lceil \sqrt{(n/K \log n)} \rceil^2$, where $c(n) \geq 1$ for all $n \in \mathbb{N}$. From Theorem 2.2(i), when we distribute $n/\log n$ targets into b^2 cells, w.h.p. there are at most $c(n) O\left(\frac{\log n}{\log \log n}\right)$ targets in any given cell. From Theorem 2.2(iv), w.h.p. there are at least $c(n)\Omega(\log n)$ agents in each cell. Thus, w.h.p, there are more agents than targets in every cell. Thus after running the ROLE ASSGMT algorithm, every target in each cell will be assigned. The maximum matching can be found in $O(1)$ time. Since each cells area is $\leq r_{\text{comm}}^2/5$,



(a) Initial agent and target positions. (b) Agent positions mid execution. (c) Final agent reaching its target.

Fig. 7. Simulation of the ETSP ASSGMT algorithm for 20 agents in a sparse environment. Targets are black dots and agents are blue squares. The ETSP tour is shown connecting the targets, and a red line is drawn agents within communication range.

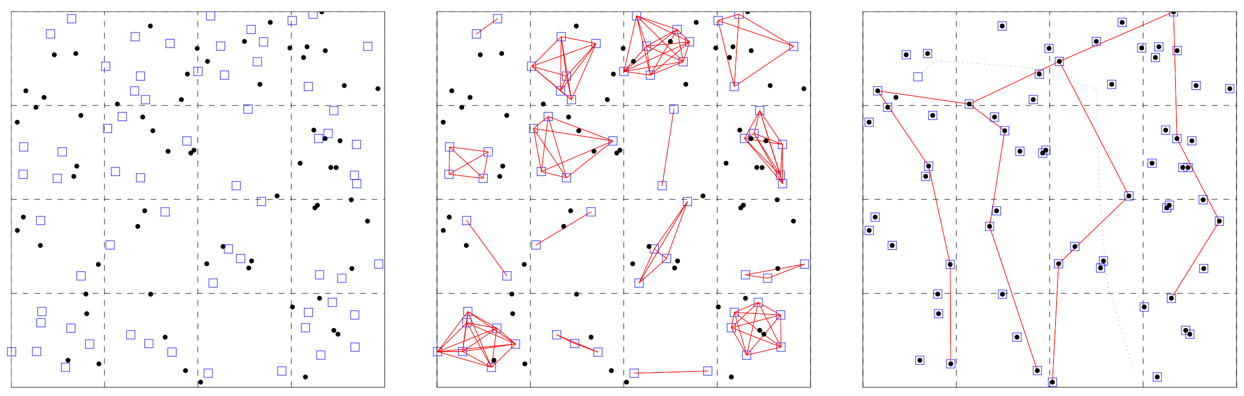
and the agents move at constant speed, the assignment will be complete in $O(1)$ time, with high probability. ■

VII. SIMULATIONS

We have performed extensive simulations of the ETSP ASSGMT and GRID ASSGMT algorithms. The ETSP ASSGMT algorithm has been simulated in both two and three dimensional environments. To compute the ETSP tour we have used the `concorde` TSP solver.[5] A representative simulation for 20 agents and targets uniformly randomly placed in a sparse environment is shown in Fig. 7. The ETSP tour is shown connecting the target positions. Dashed blue trails in Fig. 7(b) and Fig. 7(c), give the trajectories of agents that have yet to reach a target. A representative simulation of the GRID ASSGMT algorithm for 65 agents and targets uniformly randomly distributed in a dense environment is shown in Fig. 8. In Fig. 8(c) the communication between the leaders of each cell is shown with red lines, and a dashed blue trail shows the trajectory for the final agent, as it is about to reach its target in cell $C(1, 1)$.

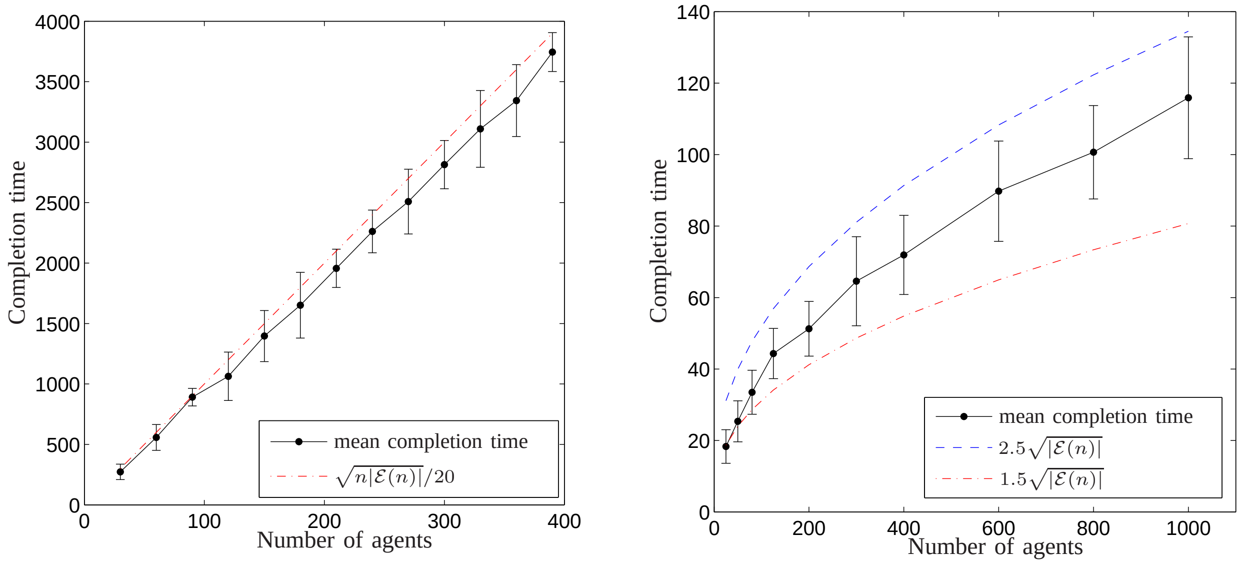
Fig. 9 contains the numerical outcomes of Monte Carlo simulations for the ETSP ASSGMT and GRID ASSGMT algorithms with uniformly randomly generated target and agent positions. Both sets of simulations were performed for agents with $r_{\text{comm}} = 10$ and $v_{\max} = 1$. Each data point is the mean completion time of 30 trials, where each trial was performed at randomly generated agent and target positions. Error bars show plus/minus one standard deviation. The simulation for the ETSP ASSGMT algorithm in Fig 9(a) was performed in a square environment with area $4r_{\text{comm}}^2 n$, and suggests that even for uniformly randomly generated positions, ETSP ASSGMT solves the target assignment problem in time proportional to $\sqrt{n|\mathcal{E}(n)|}$. The Monte Carlo simulation for the GRID ASSGMT algorithm is shown in Fig. 9(b). These simulations were performed in a square environment with area $r_{\text{comm}}^2 n/(6 \log n)$, which satisfies the bound in Theorem 6.7. For simplicity of implementation we discard trials in which there exists a cell

[5]The `concorde` TSP solver is available for research use at `http://www.tsp.gatech.edu/concorde`



(a) Initial agent and target positions. (b) Role assignment in each cell. (c) Final agent reaching target.

Fig. 8. A simulation of 65 agents in a dense environment. Targets are black disks and agents are blue squares. The partition of $\mathcal{E}(n)$ is shown in dashed lines, and red lines are drawn between communicating agents.



(a) ETSP ASSGMT algorithm in a sparse $\mathcal{E}(n)$. (b) GRID ASSGMT algorithm in a dense environment.

Fig. 9. Monte Carlo simulations for uniformly randomly generated agent and target positions. Each point is the mean completion time of 30 independent trials. Error bars show plus-minus one standard deviation.

without targets. This is justified by the fact that w.h.p. every cell contains at least one target, and thus the number of discarded trials tends to zero as n increases. The simulation suggests that asymptotically, the expected completion time is bounded below by $1.5\sqrt{|\mathcal{E}(n)|}$ and above by $2.5\sqrt{|\mathcal{E}(n)|}$. This agrees with the $O(\sqrt{|\mathcal{E}(n)|})$ bound in Theorem 6.7 and gives some idea as to the constant in front of this bound.

VIII. Extensions and Conclusions

In this paper we have attempted to present the ETSP Assgmt and Grid Assgmt algorithms in their most basic forms. In this section we discuss some extensions to these algorithms.

A. Higher Dimensional Spaces

We have presented our algorithms for the environment $\mathcal{E}(n) := [0, \ell(n)]^2 \subset \mathbb{R}^2$. However, these algorithms can be generalized to subsets of $\mathbb{R}^d$, $d \geq 1$. The ETSP Assgmt algorithm we have presented is valid for any environment $\mathcal{E}(n) \subset \mathbb{R}^d$, $d \geq 1$. In [1], we have presented time complexity bounds for environments in $\mathbb{R}^d$. In this case, the length of the ETSP tour is bounded by $O(n^{(d-1)/d}|\mathcal{E}(n)|^{1/d})$ and thus the ETSP Assgmt algorithm has time complexity in $O(n^{(d-1)/d}|\mathcal{E}(n)|^{1/d})$.[6]

The Grid Assgmt algorithm we have presented is only valid for environments in $\mathbb{R}^2$. This was done in an effort to simplify the presentation. However, the extension to $\mathbb{R}^d$ is straightforward. For example, in $\mathbb{R}^3$ the environment is partitioned into small cubes. Agents first try to find a free target in their own cube, then in their own column, then in their own plane, and then finally, they transfer into a new plane that has an available target. The worst-case bound is then given by $O(|\mathcal{E}(n)|)$, and for uniformly randomly generated target and agent positions, when the environment satisfies the bound

$$|\mathcal{E}(n)| \leq \frac{r_{\text{comm}}^2}{K(d+3)} \frac{n}{\log n},$$

where $K > 1$, is $O(|\mathcal{E}(n)|^{1/d})$, with high probability.

B. The Case of n Agents and m Targets

It should be noted that both the ETSP Assgmt and Grid Assgmt algorithms work, without any modification, when there are n agents and m targets. If $m \geq n$, at completion, then n targets are assigned and $m - n$ targets are not. When $m < n$, at completion, all m targets are assigned, and the $n - m$ unassigned agents come to a stop after losing a conflict at each of the m targets. By modifying the algorithms so that the $n - m$ unassigned agents revisit assigned targets to check for failed agents, the robustness of the algorithms can be increased. It is a straightforward exercise to alter the upper bounds when $m \neq n$. For example, the worst-case upper bound on the ETSP Assgmt algorithm becomes $O(\sqrt{|\mathcal{E}(n)|N})$, where $N := \min\{n, m\}$, and holds for any n and m. Similarly, the worst-case upper bound on the Grid Assgmt algorithm remains $O(|\mathcal{E}(n)|)$ and holds for any n and m. In addition the lower bound on the monotonic class can be easily extended when $m \geq n$. However, the extension for $m < n$ appears to require a different construction of worst-case agent and target positions.

C. Alternate Scaling Laws

We have given complexity bounds for the case when r_{comm} and v_{max} are fixed constants, and $\mathcal{E}(n)$ grows with n. We allow the environment $\mathcal{E}(n)$ to grow with n so that, as more agents are involved in the task, their workspace

[6]Here $|\mathcal{E}(n)|$ denotes the d-dimensional volume of $\mathcal{E}(n)$.

is larger. An equivalent setup would be to consider a fixed size environment, and allow r_{comm} and $v_{\max}$ to decrease with increasing n. Scaling the communication radius inversely with the number of agents arises in the study of wireless networks [22]. As the density of wireless nodes in a fixed area increase, the effects of wireless congestion and media access problems become more prevalent. To reduce these effects, the nodes reduce their transmission radius, thus reducing their interference footprint. The idea of scaling the agents' maximum speed inversely with n occurs due to physical congestion [21]. As the density of robots increases, it necessarily takes longer for the robots to travel across their environment. Motivated by this discussion, we introduce a new set of parameters, $\tilde{\mathcal{E}}$, $\tilde{r}_{\text{comm}}(n)$, and $\tilde{v}_{\max}(n)$ satisfying $|\tilde{\mathcal{E}}| \in \mathbb{R}_{>0}$ and $\tilde{v}_{\max}(n) = \Theta(\tilde{r}_{\text{comm}}(n))$. Since $\tilde{v}_{\max}(n)$ and $\tilde{r}_{\text{comm}}(n)$ scale at the same rate, the amount of time required to travel a distance $\tilde{r}_{\text{comm}}(n)$ is independent of n. Then, analogous to the definition of environment size, we define the communication range to be: *sparse* if $\tilde{r}_{\text{comm}}(n)\sqrt{n} \to 0^+$, as $n \to +\infty$; *critical* if $\tilde{r}_{\text{comm}}(n)\sqrt{n} \to \text{const} \in \mathbb{R}_{>0}$ as $n \to +\infty$; *dense* if $\tilde{r}_{\text{comm}}(n)\sqrt{n} \to +\infty$, as $n \to +\infty$.

With these definitions we can summarize the worst-case results as follows.

Corollary 8.1 (Scaling radius and speed): Consider any initial positions of n agents, with communication range $\tilde{r}_{\text{comm}}(n)$ and maximum speed $\tilde{v}_{\max}(n) = \Theta(\tilde{r}_{\text{comm}}(n))$, and n targets in the fixed environment $\tilde{\mathcal{E}}$. Then:

(i) the ETSP ASSGMT algorithm solves the target assignment problem in $O(\sqrt{n}/\tilde{r}_{\text{comm}}(n))$ time;

(ii) if $\tilde{r}_{\text{comm}}(n)$ is sparse or critical, then ETSP ASSGMT is within a constant factor of the optimal monotonic algorithm for worst-case initial conditions;

(iii) the GRID ASSGMT algorithm solves the target assignment problem in $O(1/\tilde{r}_{\text{comm}}(n)^2)$ time; and

(iv) if $\tilde{r}_{\text{comm}}(n)$ is dense or critical, then the GRID ASSGMT algorithm is within a constant factor of the optimal monotonic algorithm for worst-case initial conditions.

D. Conclusions

In this paper we have studied a version of the target assignment problem in which each agent has a list of the target positions, but has only limited communication capabilities. We introduced the class of monotonic algorithms for approaching these problems and gave a lower bound on its asymptotic performance. We introduced two algorithms in this class, the ETSP ASSGMT algorithm and the GRID ASSGMT algorithm. We have shown that in sparse environments, where communication between agents is infrequent, the ETSP ASSGMT algorithm is within a constant factor of the optimal monotonic algorithm for worst-case initial conditions. On the other hand, in dense environments, where communication is more prevalent, the GRID ASSGMT algorithm is within a constant factor of the optimal monotonic algorithm for worst-case initial conditions. Both algorithms extend to higher dimensional spaces and to problems where the number of agents and targets differ, and the GRID ASSGMT algorithm can be implemented in a sensor based version, where each agent has no knowledge of target positions, but has a limited range sensor.

There are many future research directions such as extensions to vehicles with motion constraints, or to the case when targets are dynamically appearing and disappearing. Also, we believe it is possible to extend our algorithms and analysis from the synchronous communication model to an asynchronous, or event-based, model. Another area of future research is to develop a communication framework for robotic networks that adequately models congestion and media access problems that are inherently present in wireless communications.

REFERENCES

[1] S. L. Smith and F. Bullo, "Target assignment for robotic networks: Asymptotic performance under limited communication," in *American Control Conference*, New York, Jul. 2007, pp. 1155–1160.

[2] ——, "Target assignment for robotic networks: Worst-case and stochastic performance in dense environments," in *IEEE Conf. on Decision and Control*, New Orleans, LA, Dec. 2007, pp. 3585–3590.

[3] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, 3rd ed., ser. Algorithmics and Combinatorics. Springer, 2005, no. 21.

[4] R. Burkard, "Selected topics on assignment problems," *Discrete Applied Mathematics*, vol. 123, pp. 257–302, 2002.

[5] J. E. Hopcroft and R. M. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs," *SIAM Journal on Computing*, vol. 2, no. 4, pp. 225–231, 1973.

[6] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics*, vol. 2, pp. 83–97, 1955.

[7] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, 1997.

[8] B. P. Gerkey and M. J. Mataric, "A formal analysis and taxonomy of task allocation in multi-robot systems," *International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004.

[9] M. F. Godwin, S. Spry, and J. K. Hedrick, "Distributed collaboration with limited communication using mission state estimates," in *American Control Conference*, Minneapolis, MN, Jun. 2006, pp. 2040–2046.

[10] M. Alighanbari and J. P. How, "Robust decentralized task assignment for cooperative UAVs," in *AIAA Conf. on Guidance, Navigation and Control*, Keystone, CO, Aug. 2006.

[11] C. Schumacher, P. R. Chandler, S. J. Rasmussen, and D. Walker, "Task allocation for wide area search munitions with variable path length," in *American Control Conference*, Denver, CO, 2003, pp. 3472–3477.

[12] B. J. Moore and K. M. Passino, "Distributed task assignment for mobile agents," *IEEE Transactions on Automatic Control*, vol. 52, no. 4, pp. 749–753, 2007.

[13] D. A. Castañón and C. Wu, "Distributed algorithms for dynamic reassignment," in *IEEE Conf. on Decision and Control*, Maui, HI, Dec. 2003, pp. 13–18.

[14] G. Arslan, J. R. Marden, and J. S. Shamma, "Autonomous vehicle-target assignment: A game theoretic formulation," *ASME Journal on Dynamic Systems, Measurement, and Control*, vol. 129, no. 5, pp. 584–596, 2007.

[15] M. M. Zavlanos and G. J. Pappas, "Dynamic assignment in distributed motion planning with local coordination," *IEEE Transactions on Robotics*, vol. 24, no. 1, pp. 232–242, Feb. 2008.

[16] K. J. Supowit, E. M. Reingold, and D. A. Plaisted, "The traveling salesman problem and minimum matching in the unit square," *SIAM Journal on Computing*, vol. 12, pp. 144–156, 1983.

[17] S. Arora, "Polynomial-time approximation schemes for the Euclidean TSP and other geometric problems," *Journal of the ACM*, vol. 45, no. 5, pp. 753–782, 1998.

[18] R. Motwani and P. Raghavan, *Randomized Algorithms*. Cambridge University Press, 1995.

[19] F. Xue and P. R. Kumar, "The number of neighbors needed for connectivity of wireless networks," *Wireless Networks*, vol. 10, no. 2, pp. 169–181, 2004.

[20] M. Penrose, *Random Geometric Graphs*, ser. Oxford Studies in Probability. Oxford University Press, 2003.

[21] V. Sharma, M. Savchenko, E. Frazzoli, and P. Voulgaris, "Transfer time complexity of conflict-free vehicle routing with no communications," *International Journal of Robotics Research*, vol. 26, no. 3, pp. 255–272, 2007.

[22] P. Gupta and P. R. Kumar, "The capacity of wireless networks," *IEEE Transactions on Information Theory*, vol. 46, no. 2, pp. 388–404, 2000.

APPENDIX A

FORMAL DESCRIPTION THE ETSP ASSGMT ALGORITHM

Algorithm 1: Initialization of agent i in ETSP ASSGMT.

```
   Assumes: Agent $i$ has the target array $\mathbf{q}^{[i]} := \mathbf{q}$, stored in its memory.
1  Compute a TSP tour of $\mathbf{q}^{[i]}$, $\mathrm{tour}(\mathbf{q}^{[i]})$, and set $\mathbf{q}^{[i]} := \mathrm{tour}(\mathbf{q}^{[i]})$.
2  Compute the closest target in $\mathbf{q}^{[i]}$, and set $\mathrm{curr}^{[i]}$ equal to its index: $\mathrm{curr}^{[i]} := \arg\min_{j\in\mathcal{I}}\{\|\mathbf{q}_j^{[i]} - \mathbf{p}^{[i]}\|\}$.
3  Set $\mathrm{next}^{[i]} := \mathrm{curr}^{[i]} + 1 \pmod{n}$.
4  Set $\mathrm{prev}^{[i]} := \mathrm{curr}^{[i]} - 1 \pmod{n}$.
5  Set $\mathrm{status}^{[i]} := \mathbf{1}_n$ (i.e., an $n$-tuple containing $n$ ones).
```

Algorithm 2: COMM-RD, executed at each communication round.

```
    Assumes: Agent $i$ has been initialized as in Algorithm 1.
1   Compute $\mathrm{dist}^{[i]} := \|\mathbf{p}^{[i]} - \mathbf{q}^{[i]}_{\mathrm{curr}^{[i]}}\|$.
2   Broadcast $\mathrm{msg}^{[i]} := (\mathrm{prev}^{[i]}, \mathrm{curr}^{[i]}, \mathrm{next}^{[i]}, i, \mathrm{dist}^{[i]})$
3   Receive $\mathrm{msg}^{[k]}$, from each $k \neq i$ satisfying $\|\mathbf{p}^{[i]} - \mathbf{p}^{[k]}\| \leq r$.
4   foreach $\mathrm{msg}^{[k]}$ received do
5       for $s = \mathrm{prev}^{[k]} + 1$ to $\mathrm{next}^{[k]} - 1 \pmod{n}$ do
6           if $s \neq \mathrm{curr}^{[i]}$ then Set $\mathrm{status}^{[i]}(s) := 0$
7       if $\mathrm{prev}^{[k]} = \mathrm{next}^{[k]} = \mathrm{curr}^{[k]} \neq \mathrm{curr}^{[i]}$ then Set $\mathrm{status}^{[i]}(\mathrm{curr}^{[k]}) := 0$
8       if $\mathrm{curr}^{[i]} = \mathrm{curr}^{[k]}$ then
9           if ($\mathrm{dist}^{[i]} > \mathrm{dist}^{[k]}$) OR ($\mathrm{dist}^{[i]} = \mathrm{dist}^{[k]}$ AND $i < k$) then
10              Set $\mathrm{status}^{[i]}(\mathrm{curr}^{[i]}) := 0$.
11          if $\mathrm{next}^{[i]} \neq \mathrm{curr}^{[i]}$ then Set $\mathrm{status}^{[i]}(\mathrm{next}^{[i]}) := 0$.
12          if $\mathrm{next}^{[k]} \neq \mathrm{curr}^{[i]}$ then Set $\mathrm{status}^{[i]}(\mathrm{next}^{[k]}) := 0$.
13  if $\mathrm{status}^{[i]}(j) = 0$ for every target $j$ then Exit ETSP ASSGMT and stop motion.
14  while $\mathrm{status}^{[i]}(\mathrm{curr}^{[i]})$=0 do $\mathrm{curr}^{[i]} := \mathrm{curr}^{[i]} + 1 \pmod{n}$.
15  Set $\mathrm{next}^{[i]} := \mathrm{curr}^{[i]} + 1 \pmod{n}$.
16  while $\mathrm{status}^{[i]}(\mathrm{next}^{[i]})$=0 do $\mathrm{next}^{[i]} := \mathrm{next}^{[i]} + 1 \pmod{n}$.
17  while $\mathrm{status}^{[i]}(\mathrm{prev}^{[i]})$=0 do $\mathrm{prev}^{[i]} := \mathrm{prev}^{[i]} - 1 \pmod{n}$.
```

APPENDIX B

FORMAL DESCRIPTION OF THE GRID ASSGMT ALGORITHM

As noted in Remark 6.4, we have simplified the presentation of the UNASSIGNED algorithm by assuming that every cell initially contains at least one agent and one target. It is straightforward to relax this assumption. If a cell has no targets, then any agents initially in the cell move to the cell below, and the empty cell is ignored for the rest of the algorithm. If there is a cell that contains targets but no agents, then the first agents to enter the cell run the ROLE ASSGMT algorithm and one becomes the leader. Agents move at speed v_{max}.

TABLE I
VARIABLES FOR THE GRID ASSGMT ALGORITHM.

Agent role	Variable	Description	Value
all	$\text{currcell}^{[i]}$	cell currently occupied by agent i	a cell $C(r,c)$
	$\text{leader}^{[i]}$	cell for which agent i is leader	a cell $C(r,c)$ or `null`
	$\text{curr}^{[i]}$	agent i's assigned target	at target in $\mathcal{Q}$, or `null`
unassigned	$\text{dircol}^{[i]}$	direction of travel in column c.	`up` or `down`.
	$\text{dirrow}^{[i]}$	direction of travel when in row 1	`left` or `right`
	$\text{colstatus}^{[i]}(c)$	records whether or not there are free targets in column c	`full` or `notfull`
	$\text{prevcell}^{[i]}$	previous cell occupied by agent i	a cell $C(r,c)$
$C(r,c)$ leader	$\text{taravail}^{[i]}(r,c)$	set of available targets in $C(r,c)$	a subset of $\mathcal{Q}$
	$\Delta^{[i]}(r,c)$	(# of targets) $-$ (# of agents) in $C(r,c)$	an integer
	$\Delta^{[i]}_{\text{blw}}(r,c)$	est. of (# of targets) $-$ (# of agents) in $C(r+1,c)$ to $C(b,c)$	an integer or $+\infty$
$C(1,c)$ leader	$\Delta^{[i]}_{\text{rght}}(1,c)$	est. of (# of targets) $-$ (# of agents) in columns $c+1$ to b	an integer or $+\infty$

Algorithm 3: ROLE ASSGMT, executed at the start of the GRID ASSGMT algorithm to assign roles, and initialize agent i.

```
   Assumes: Agent $i$ is in $C(r,c)$, knows $\mathcal{E}(n)$, and either (1) knows all target positions, or (2) has $r_{\text{sense}} \geq \sqrt{2/5}\, r_{\text{comm}}$.
1  Compute $b$ as in Lemma 2.3, partition $\mathcal{E}(n)$ into the $b^2$ square cells.
2  Set $\text{currcell}^{[i]} := C(r,c)$, $\text{leader}^{[i]} :=$ null and $\text{curr}^{[i]} :=$ null.
3  Broadcast $\text{msg}^{[i]}$ containing $\text{UID}^{[i]}$, $\mathbf{p}^{[i]}$, and $\text{currcell}^{[i]}$ to agents in $\text{currcell}^{[i]}$.
4  Receive $\text{msg}^{[k]}$ from, all agents in $C(r,c)$.
5  Use the MAXIMAL MATCH algorithm to find a maximum matching between agents and targets $C(r,c)$.
6  Elect a leader among assigned agents in $C(r,c)$.
7  case unassigned
8      Set $\text{dircol}^{[i]} :=$ down, $\text{dirrow}^{[i]} :=$ right, and $\text{colstatus}^{[i]}(c)$ to notfull for each $c \in \{1,\dots,b\}$.
9      Run UNASSIGNED algorithm.
10 case assigned to $\mathbf{q} \in \mathcal{C}(r,c)$ and not elected leader
11     Set $\text{curr}^{[i]} := \mathbf{q}$, and move to $\text{curr}^{[i]}$ at speed $v_{\text{max}}$
12 case assigned and elected leader
13     Set $\text{leader}^{[i]} := \text{currcell}^{[i]}$, $\text{curr}^{[i]} := \mathbf{q}$, and move to $\text{curr}^{[i]}$ at speed $v_{\text{max}}$
14     Set $\Delta^{[i]}(r,c)$ to number of targets in $C(r,c)$ minus number of agents in $C(r,c)$.
15     Set $\text{taravail}^{[i]}(r,c)$ to the collection of unassigned targets in $C(r,c)$.
16     Set $\Delta^{[i]}_{\text{blw}}(r,c)$ to $+\infty$ if $r \in \{1,\dots,b-1\}$ and to 0 if $r = b$.
17     if $r = 1$ then Set $\Delta^{[i]}_{\text{rght}}(c)$ to $+\infty$ if $c \in \{1,\dots,b-1\}$ and to 0 if $c = b$.
18     Run LEADER algorithm.
```

Algorithm 4: LEADER, executed at each communication round.

```
   Assumes: Agent $i$ is the leader of $C(r,c)$.
1  Send $\mathrm{msg}_1^{[i]} := \Delta_{\mathrm{blw}}^{[i]}(r,c) + \Delta^{[i]}(r,c)$ to leader in cell $C(r-1,c)$.
2  if $r = 1$ then
3      Send $\mathrm{msg}_2^{[i]} := \Delta_{\mathrm{rght}}^{[i]}(c) + \Delta_{\mathrm{blw}}^{[i]}(1,c) + \Delta^{[i]}(1,c)$ to leader of $C(1,c-1)$.
4      Receive $\mathrm{msg}_2^{[k]}$ from leader $k$ of $C(1,c+1)$ and set $\Delta_{\mathrm{rght}}^{[i]}(c) := \mathrm{msg}_2^{[k]}$.
5      For each enter msg from an agent coming from $C(1,c+1)$, add 1 to $\Delta_{\mathrm{rght}}^{[i]}(1,c)$ and for each exit msg from an
       agent going to $C(1,c+1)$ subtract 1 from $\Delta_{\mathrm{rght}}^{[i]}(1,c)$.
6      If $\Delta_{\mathrm{rght}}^{[i]}(1,c) > 0$ and an enter msg was received from an agent coming from $C(1,c+1)$, then set $\Delta_{\mathrm{rght}}^{[i]}(1,c) := 0$.
7  Receive $\mathrm{msg}_1^{[k]}$ from leader $k$ of $C(r+1,c)$, and set $\Delta_{\mathrm{blw}}^{[i]}(r,c) := \mathrm{msg}_1^{[k]}$.
8  Subtract 1 from $\Delta^{[i]}(r,c)$ for each enter msg received, and add 1 for each exit msg received.
9  For each enter msg from an agent coming from $C(r+1,c)$, add 1 to $\Delta_{\mathrm{blw}}^{[i]}(r,c)$ and for each exit msg from an agent
   going to $C(r+1,c)$ subtract 1 from $\Delta_{\mathrm{blw}}^{[i]}(r,c)$.
10 If $\Delta_{\mathrm{blw}}^{[i]}(r,c) > 0$ and an enter msg was received from an agent coming from $C(r+1,c)$, then set $\Delta_{\mathrm{blw}}^{[i]}(r,c) := 0$.
11 forall queries on availability of target in $C(r,c)$ do
12     if $\mathrm{taravail}^{[i]} \neq \emptyset$ then
13         Select a target in $\mathrm{taravail}^{[i]}$, assign it requesting agent, and remove it from $\mathrm{taravail}^{[i]}$.
14     else if $\mathrm{taravail} = \emptyset$ then Reply no.
15 forall queries on availability of target below $C(r,c)$ do
16     Respond yes to $\Delta_{\mathrm{blw}}^{[i]}(r,c)$ requests, and no to all others.
17 if $r = 1$ then
18     forall queries on availability of target to right of column $c$ do
19         Respond yes to $\Delta_{\mathrm{rght}}^{[i]}(c)$ requests, and no to all others.
```

Algorithm 5: UNASSIGNED, executed each time a new cell is entered.

```
Assumes: Agent $i$ has run ROLE ASSGMT, and currcell$^{[i]} = C(r,c)$.
1  Query leader of $C(r,c)$ on free targets in currcell$^{[i]}$.
2  if leader returns a target $\mathbf{q} \in C(r,c)$ then  Set curr$^{[i]} := \mathbf{q}$, and move to target.
3  else if leader returns no then
4      case dircol$^{[i]}$ = down
5          Query leader on availability of target below $C(r,c)$.
6          if leader returns yes then
7              Set prevcell$^{[i]}$ := currcell$^{[i]}$ and currcell$^{[i]} := C(r+1,c)$
8          else if leader returns no then
9              Set dircol$^{[i]}$ := up, prevcell$^{[i]}$ := currcell$^{[i]}$ and currcell$^{[i]} := C(r-1,c)$.
10     case (dircol$^{[i]}$ = up) and $(r > 1)$
11         Set prevcell$^{[i]}$ := currcell$^{[i]}$, currcell$^{[i]} := C(r-1,c)$, and dircol$^{[i]}$ := up
12     case (dircol$^{[i]}$ = up) and $(r = 1)$ and (dirrow$^{[i]}$ = right)
13         Set colstatus$^{[i]}(c)$ := full.
14         Query leader on availability to the right of column $c$.
15         if leader returns yes then
16             Set prevcell$^{[i]}$ := currcell$^{[i]}$, currcell$^{[i]} := C(1,c+1)$
17             if colstatus$^{[i]}(c+1)$ = notfull then  dircol$^{[i]}$ := down.
18         else if leader returns no then
19             Set prevcell$^{[i]}$ := currcell$^{[i]}$, currcell$^{[i]} := C(1,c-1)$, dirrow$^{[i]}$ := left.
20             Set colstatus$^{[i]}(c^*)$ := full for each $c^* \in \{c+1,\dots,b\}$.
21             if colstatus$^{[i]}(c-1)$ = notfull then  dircol$^{[i]}$ := down.
22     case (dircol$^{[i]}$ = up) and $(r = 1)$ and (dirrow$^{[i]}$ = left)
23         Set prevcell$^{[i]}$ := currcell$^{[i]}$ and currcell$^{[i]} := C(1,c-1)$.
24         if colstatus$^{[i]}(c-1)$ = notfull then  dircol$^{[i]}$ := down.
25     Send exit to leader in prevcell$^{[i]}$, enter to leader in currcell$^{[i]}$, and move to currcell$^{[i]}$.
```