Autonomous Robots (2020) 44:43–55
https://doi.org/10.1007/s10514-019-09849-0

Optimizing multi-robot communication under bandwidth constraints

Ryan J. Marcotte[1] · Xipeng Wang[1] · Dhanvin Mehta[1] · Edwin Olson[1]

Received: 23 August 2018 / Accepted: 20 March 2019 / Published online: 17 April 2019

Abstract
Robots working collaboratively can share observations with others to improve team performance, but communication bandwidth is limited. Recognizing this, an agent must decide which observations to communicate to best serve the team. Accurately estimating the value of a single communication is expensive; finding an optimal combination of observations to put in the message is intractable. In this paper, we present OCBC, an algorithm for Optimizing Communication under Bandwidth Constraints. OCBC uses forward simulation to evaluate communications and applies a bandit-based combinatorial optimization algorithm to select what to include in a message. We evaluate OCBC's performance in a simulated multi-robot navigation task. We show that OCBC achieves better task performance than a state-of-the-art method while communicating up to an order of magnitude less.

Keywords Communication decision-making · Multi-robot systems · Multi-robot planning

1 Introduction

Robots that collaborate on tasks can share information about their environment, helping their teammates make better decisions. When wireless bandwidth is limited, however, a robot may not be able to communicate all its observations. How should a robot decide what to communicate when faced with such a constraint?

Answering this question requires estimating the value of a proposed communication. A direct approach is to predict the effect of the message on team reward. However, such a direct approach is intractable for many standard multi-agent models. For example, deciding a Dec-MDP or Dec-POMDP with communication is NEXP-complete (Pynadath and Tambe 2002; Goldman and Zilberstein 2004). As a consequence, most direct methods are limited to problems with only a few agents, states, and actions (Roth et al. 2005; Carlin and Zilberstein 2009a, b).

This complexity motivated indirect methods of estimating communication value. For example, some methods measure the information content contained in a message and assign high value to information-rich messages (Williamson et al. 2008, 2009). Others track the coherence of the ego-agent's internal models of its teammates and communicate to keep the team's beliefs consistent (Best et al. 2018b; Wu et al. 2011; Roth et al. 2005).

In this paper, we present a direct yet tractable method for reasoning about communication actions. To enable this tractability, we employ a specialized model of multi-agent decision-making. This modified Dec-MDP model (c.f. Becker et al. 2004; Unhelkar and Shah 2016) makes assumptions that aid in tractability while maintaining fidelity to real-world problems. It features a deterministic transition function that is initially unknown to the agents, which corresponds to the unknown map in exploration-style robotics tasks. Furthermore, the model is factored; that is, each agent makes decisions independently. Factored problems occur whenever agents collaborate while working on individual sub-tasks.

These features of the model make it tractable for an agent to evaluate a communication decision in terms of its expected effect on reward. Because the model is factored, an agent can

This is one of the several papers published in *Autonomous Robots* comprising Special Issue on Robot Communication Challenges: Real-World Problems, Systems, and Methods.

✉ Ryan J. Marcotte
ryanjmar@umich.edu

Xipeng Wang
xipengw@umich.edu

Dhanvin Mehta
dhanvinm@umich.edu

Edwin Olson
ebolson@umich.edu

[1] University of Michigan, Ann Arbor, MI, USA

compute its action policy independently of its teammates. Agents can use fast incremental algorithms (e.g. Koenig and Likhachev 2005) to plan through the state-action space since the transition function is deterministic. Quick, independent planning allows an agent to quickly forward simulate the behavior of its teammates. This leads to a key insight of this paper: fast forward simulations make it tractable to estimate the effect of a message on team reward. Our first contribution then is a method that uses such forward simulations to evaluate messages directly in reward space.

Even once an agent has assigned a value to candidate messages, it still needs a mechanism for making communication decisions. Most existing methods (e.g. Wu et al. 2011; Becker et al. 2009; Unhelkar and Shah 2016) assume that the content of a message is static and consists of, for example, the agent's entire observation history. They estimate the value of this message, then weigh that value against a fixed communication cost that is part of the problem's reward structure. Such an approach answers the question of *when* the agent should communicate.

To respect practical bandwidth constraints, however, a robot must reason about *what* to communicate, a problem that has received little attention (Roth et al. 2006). Consider for example a scenario in which a robot makes observations more quickly than the network would support sharing them. The robot then needs to decide which observations are worthwhile to send.

Our second contribution addresses this challenge. Namely, we apply a bandit-based combinatorial optimization algorithm (Chen et al. 2014) to select observations to communicate. This algorithm estimates the expected reward distribution associated with communicating each observation by repeatedly forward simulating the outcome of messages containing that observation. It uses these estimates to build an approximately optimal message.

We call our complete method OCBC (Optimizing Communication under Bandwidth Constraints). We evaluate the performance of OCBC in a simulated multi-robot navigation problem. We compare it to a state-of-the-art approach (Unhelkar and Shah 2016) and show that OCBC achieves better task performance with less communication.

2 Preliminaries

2.1 Model formulation and application

Consider the task of multiple robots navigating in an unknown environment as illustrated in Fig. 1. The robots all move toward a goal destination, and they observe their environment as they travel through it. The robots can share these observations with their teammates to help them reach the destination more quickly.

We will use this example task to explain our model formulation.

2.1.1 Agent states

We denote the team of n agents as $I = \{1, \ldots, n\}$. At each time step $t \in \{1, \ldots, T\}$, agent i is in state $s_i(t) \in S$ and takes action $a_i(t) \in A$. In our example task, each robot's state is its current cell in the map, and its available actions are to move in any one of the cardinal directions (i.e. $A = \{\mathrm{N, E, S, W}\}$)

Each agent has some goal state s_i^{GOAL} that it tries to reach, and each agent knows the desired joint goal state $\boldsymbol{s}^{\mathrm{GOAL}} = \{s_1^{\mathrm{GOAL}}, \ldots, s_n^{\mathrm{GOAL}}\}$. Each agent i knows its own state s_i but

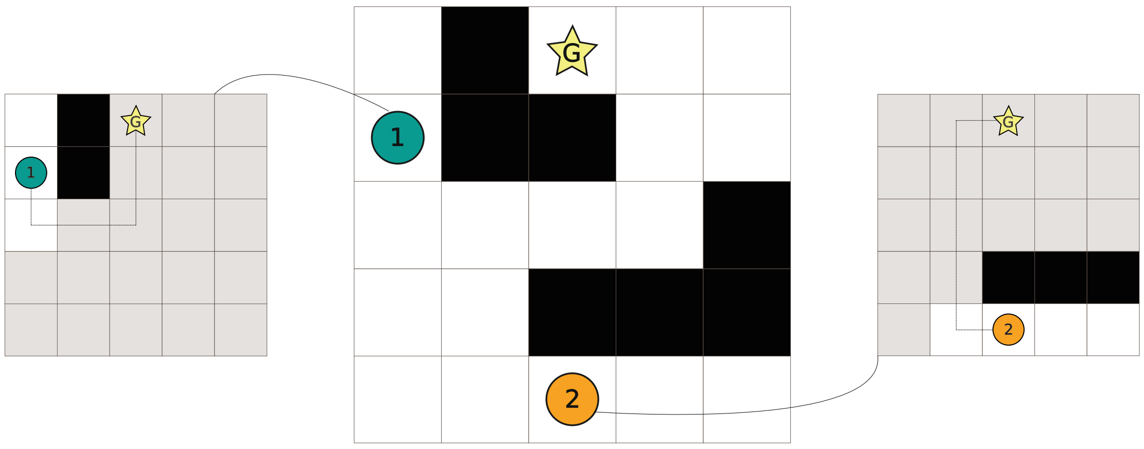


Fig. 1 An example problem in which multiple robots independently navigate through an unknown environment. All robots try to reach the goal as quickly as possible; team reward depends on the total time taken by all robots. A robot observes the occupancy of adjacent cells as it moves through its environment. The robot can communicate these observations to its teammates to help them complete the task more quickly

cannot directly observe the joint state $\boldsymbol{s} = \{s_1, \dots, s_n\}$. In our example task, this corresponds to the robot having a reliable localization system that provides its current location but not having a way to observe the locations of its teammates.

2.1.2 State transition function

The state transition function is deterministic (i.e. $P(s, a, s') \in \{0, 1\}$ for $s, s' \in S$ and $a \in A$), which corresponds to a robot having reliable closed-loop actuation. In this case, the success of an action depends only on whether the destination cell is occupied. Tasks like this have a spatial relationship between states and actions, giving the transition function two characteristics we exploit in this paper.

First, a significant portion of the transition function is trivially zero-valued. Specifically, $P(s, a, s') = 0$ for any pair of states s, s' that do not correspond to adjacent cells or any actions a that would not move the robot between the two cells.

Second, many remaining elements of the transition function have a well-defined relationship. Specifically, all (non-trivial) elements transitioning to state s' are equal in value. We denote this value as $x_{s'} = P(s, a, s') \in \{0, 1\}$ for any state s and action a that would transition the robot to successor state s' if the cell at s' were unoccupied. This value is the same for any such state-action pair.

To better understand the spatial structure of the transition function and the significance of the value $x_{s'}$, consider the following 2×2 grid environment:

0	1
2	3

.

Transition function elements such as $P(0, \cdot, 3)$ are trivially zero-valued since no single action could move the robot from cell 0 to cell 3. Likewise, an element such as $P(0, \mathrm{W}, 2)$ is trivially zero-valued since the action W cannot move the robot from cell 0 to cell 2. Of the non-trivial elements of the transition function, all elements transitioning to a given state will have the same value. For example, we have $x_1 = P(0, \mathrm{E}, 1) = P(3, \mathrm{N}, 1) = 0$ and $x_2 = P(0, \mathrm{S}, 2) = P(3, \mathrm{W}, 2) = 1$. In other words, the elements of the transition function that would lead to cell 1 have value 0 since cell 1 is occupied, and those that would lead to cell 2 have value 1 since cell 2 is unoccupied. From this, observe that the value x_s corresponds to the occupancy of the cell at state s. For the remainder of the paper, agents maintain the x_s values in place of the transition function. The full transition function can be constructed from these values when needed.

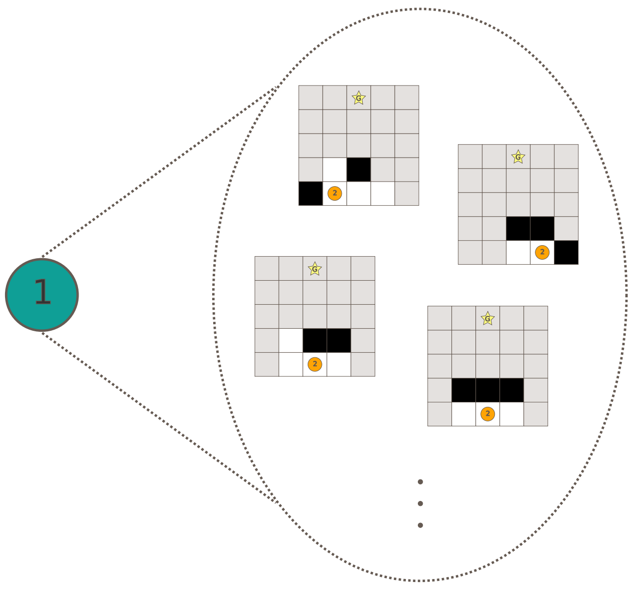


Fig. 2 An illustration of Robot 1's belief over models of its teammate (Robot 2) in the multi-robot navigation task. These models represent possible locations and maps for Robot 2

2.1.3 Transition beliefs

The agents do not know the true state transition function P. Equivalently, the agents do not know the true value of x_s for $s \in S$. Instead, each agent i maintains a belief $b_i(x_s) \in [0, 1]$ for each x_s of the transition function. We denote the belief of agent i over all x_s as $b_i(x)$.

In our concrete example, the robots do not know the true map of the environment (i.e. P). The map is given by the occupancy value of its cells (i.e. x_s). The belief $b_i(x_s)$ corresponds to believed likelihood that a cell is occupied, which we assume is independent of any other $b_i(x_{s'})$. We illustrate such a belief in Fig. 1.

2.1.4 Teammate beliefs

Because an agent cannot directly observe the states or transition beliefs of other agents, it must represent these values with a belief distribution. Specifically, each agent i maintains a distribution $b_i(m_j)$ over possible models for each other agent j. An agent model $m_j = (s_j, s_j^{\mathrm{GOAL}}, b_j(x))$ consists of a state, goal state, and transition belief.

In our example task, an agent model corresponds to a single estimate m_j of another robot's location and map. The belief $b_i(m_j)$ is a belief over all possible models, which we approximate with a particle filter (see Sect. 3.4). We illustrate such a belief in Fig. 2.

We denote the ego-agent's beliefs about all its teammates as $b_i(m)$.

2.1.5 Action policies

All agents select actions using the same deterministic planning algorithm. This planner produces a policy $\pi_i : S \to A$ for agent i given its current state s_i, its goal state s_i^{GOAL}, and its transition belief $b_i(x)$.

In our example task, robots compute the shortest path between their location and the goal location using their current map, assuming that unobserved cells are unoccupied (Koenig and Likhachev 2005).

2.1.6 Observations

An observation $\omega^{(s)} \in \{0, 1\}$ is sampled from the observation function $O(\omega^{(s)}|x_s)$. In our example task, this corresponds to an observation of the occupancy of the cell at state s. Though our formulation admits a stochastic observation function, we assume in our evaluation that O is deterministic, in which case observation $\omega^{(s)}$ reveals the true value of x_s.

At each time step t, agent i receives a set of observations $\boldsymbol{\omega}_i(t)$. This corresponds in our example task to a robot observing all its adjacent cells.

2.1.7 Communication

Agent i carries out a communication action $c_i(t) \in C$ at each time step t. This broadcasts observations $\boldsymbol{\omega}_{c_i}$ to all other agents along with the ego-agent's state s_i. This results in the other agents incorporating the associated observations into their beliefs.

2.1.8 Reward

We compute each agent's reward independently and then combine the individual rewards to yield the team reward (i.e. the reward function is factored). An agent's reward comes from two components. The action reward function $R_A : S \times A \to \mathbb{R}$ gives the reward an agent receives after taking an action from a particular state. The communication reward function $R_C : C \to \mathbb{R}$ assigns reward based on a communication action. The total reward for the team, γ, is given by

$$\begin{aligned}\gamma &= \gamma_A + \gamma_C \\ &= \sum_{i \in I} \gamma_{A\,i} + \gamma_{C\,i} \\ &= \sum_{i \in I} \sum_{t \in T} R_A\big(s_i(t), a_i(t)\big) + R_C\big(c_i(t)\big).\end{aligned} \tag{1}$$

In our example task, we assign action reward -1 for each step the robots take until they reach the goal. We assign communication reward differently depending on the communication paradigm, which we discuss next.

2.2 Communication paradigms

Roth et al. (2006) introduce the concept of communication paradigms, which are sets of rules governing how much communication is allowed or how much communication costs. They propose three such paradigms, which we summarize here. These paradigms give context to OCBC and related methods. In Sect. 4, we use these paradigms to structure our evaluation and facilitate comparison with existing work.

2.2.1 Fixed-cost communication

Most decision-theoretic methods (e.g. Unhelkar and Shah 2016; Carlin and Zilberstein 2009b; Wu et al. 2011) only focus on the question of *when* to communicate. This limits the set of possible communication actions C to only two members: share all information possible (e.g. the ego-agent's entire observation history) or do not communicate at all. The former action is assigned a fixed cost (i.e. $R_C(c) = \epsilon < 0$) while the latter has zero cost. We call this paradigm *Fixed-Cost Communication*.

2.2.2 Proportional-cost communication

To penalize excessive bandwidth consumption, communication cost should depend on message size. This is the *Proportional-Cost Communication* paradigm, in which the cost of a communication action is proportional to the length of the associated message (i.e. $R_C(c) \propto |\boldsymbol{\omega}_c|$). Roth et al. (2006) use this paradigm to motivate the question of *what* to communicate.

2.2.3 Fixed-bandwidth communication

Multi-robot teams using a wireless network must share some fixed amount of bandwidth (Bianchi 1998; Hiertz et al. 2007; Shrader and Ephremides 2007). We denote the amount of bandwidth available to the team at each time step as β. The team allocates a portion of this bandwidth $\beta_i(t)$ to each agent i at time step t through an offline round robin schedule or other similar mechanism. Each agent must then decide how to use this bandwidth allocation. We call such a paradigm *Fixed-Bandwidth Communication*.

In the *Fixed-Bandwidth* paradigm, there is no cost associated with a communication action (i.e. $R_C(\cdot) = 0$). The set of communication actions available to an agent, $C_i(t)$, contains all messages that fit within its bandwidth allocation at time step t. That is, $C_i(t) = \{c : |\boldsymbol{\omega}_c| \le \beta_i(t)\}$.

As we present it in the following section, OCBC fits within the *Fixed-Bandwidth* paradigm. In Sect. 4, we will discuss how to modify OCBC to compare its performance to methods from the other two paradigms.

3 Approach

We now introduce OCBC and its supporting algorithms. Section 3.1 explains OCBC, which is a bandit-based approach to optimizing multi-robot communication under bandwidth constraints. Section 3.2 shows how OCBC evaluates communication actions using forward simulation. These two sections correspond to our two primary contributions.

The remaining two sections discuss how we implement OCBC in the context of a sequential decision-making agent. Section 3.3 gives the main sequential decision-making method, and Sect. 3.4 deals with maintaining and updating agent beliefs within that method.

3.1 Optimizing communication under bandwidth constraints (OCBC)

An agent typically has multiple observations it could communicate with its teammates. When the agent's bandwidth allocation is too small to share all of those observations, it must decide which observations to include. This requires estimating the reward associated with each observation, which we do by forward simulation (see Sect. 3.2). Because of the uncertainty in the ego-agent's beliefs, though, any one forward simulation only samples from the distribution of rewards for a given communication. In this way, the observations are like levers a multi-armed bandit could pull, and our goal is to efficiently identify the observations with the highest expected reward.

To this end, we apply the CSAR Algorithm (Chen et al. 2014), which is a bandit-based method that performs combinatorial optimization under uncertain rewards. CSAR tries to find an optimal member of a specified decision class (e.g. all arm combinations of a certain size). CSAR stands for "Combinatorial Successive Accept Reject", a title that explains the algorithm's basic function. Given a set of K arms, CSAR makes K successive decisions to either accept or reject an arm. In between decisions, CSAR samples from the remaining arms to better estimate their associated reward distributions. CSAR makes the accept-reject decision on the arm for which it has the highest-confidence reward distribution estimate. By the end of this process, the set of accepted arms is an approximately optimal member of the decision class. See (Chen et al. 2014) for details on suboptimality bounds of the CSAR method.

Algorithm 1 Optimizing Communication under Bandwidth Constraints (OCBC)

```
    For convenience, let $\beta$, $b_i(x)$, $b_i(m)$, $\hat{\Delta}_k$, and $\mu_k$ be global variables
    shared among all functions
 1: function OPTIMIZECOMMUNICATION($b_i(x), b_i(m), \boldsymbol{\omega}$)
 2:     $K \leftarrow |\boldsymbol{\omega}|$
 3:     $\boldsymbol{\omega}^{\text{CAND}} \leftarrow \boldsymbol{\omega}$; $\boldsymbol{\omega}^{\text{ACC}} \leftarrow \emptyset$
 4:     $\hat{\Delta}_k \leftarrow \emptyset$; $\mu_k \leftarrow 0,\ k \in \{1, \dots, K\}$
 5:     for $k \leftarrow 1, \dots, K$ do
 6:         $B_k \leftarrow$ ALLOCATESAMPLEBUDGET($B, K, k$)
 7:         ESTIMATEREWARDDISTRIBUTIONS($B_k, \boldsymbol{\omega}^{\text{CAND}}$)
 8:         $\hat{\boldsymbol{\omega}}^* \leftarrow$ ASSEMBLE($\boldsymbol{\mu}, \boldsymbol{\omega}^{\text{ACC}}, \boldsymbol{\omega}^{\text{CAND}}$)
 9:         $l \leftarrow$ SELECT($\boldsymbol{\omega}^{\text{ACC}}, \boldsymbol{\omega}^{\text{CAND}}, \hat{\boldsymbol{\omega}}^*$)
10:         if $\omega_l \in \hat{\boldsymbol{\omega}}^*$ then
11:             $\boldsymbol{\omega}^{\text{ACC}} \leftarrow \boldsymbol{\omega}^{\text{ACC}} \cup \{\omega_l\}$
12:         $\boldsymbol{\omega}^{\text{CAND}} \leftarrow \boldsymbol{\omega}^{\text{CAND}} \setminus \{\omega_l\}$
13:     return $\boldsymbol{\omega}^{\text{ACC}}$

14: function ALLOCATESAMPLEBUDGET($B, K, k$)
15:     $\kappa(y) \triangleq \sum_{y=1}^{K} \frac{1}{y}$
16:     $f(y) \triangleq \lceil \frac{B-K}{\kappa(y)(K-y+1)} \rceil, \quad y > 0$
17:     $f(0) \triangleq 0$
18:     return $f(k) - f(k-1)$

19: function ESTIMATEREWARDDISTRIBUTIONS($B_k, \boldsymbol{\omega}^{\text{CAND}}$)
20:     for $1, \dots, B_k$ do
21:         $\tilde{\boldsymbol{m}} \leftarrow \bigcup_{j \in I \setminus i}$ Sample $m_j \sim b_i(m_j)$
22:         Sample $\tilde{x} \sim b_i(x)$
23:         $\tilde{P} \leftarrow$ CONSTRUCTTRANSITIONFUNCTION($\tilde{x}$)
24:         for $\omega_l \in \boldsymbol{\omega}^{\text{CAND}}$ do
25:             $\hat{\delta} \leftarrow$ EVALUATECOMMUNICATION($\{\omega_l\}, \tilde{\boldsymbol{m}}, \tilde{P}$)
26:             $\hat{\Delta}_l \leftarrow \hat{\Delta}_l \cup \{\hat{\delta}\}$
27:             $\mu_l \leftarrow \frac{1}{|\hat{\Delta}_l|} \sum_{\hat{\delta} \in \hat{\Delta}_l} \hat{\delta}$

28: function ASSEMBLE($\boldsymbol{\mu}, \boldsymbol{\omega}^{\text{ACC}}, \boldsymbol{\omega}^{\text{CAND}}$)
29:     $\hat{\boldsymbol{\omega}}^* \leftarrow \boldsymbol{\omega}^{\text{ACC}}$
30:     while $|\hat{\boldsymbol{\omega}}^*| < \beta_i$ and $|\boldsymbol{\omega}^{\text{CAND}}| > 0$ do
31:         $l \leftarrow \arg\max_{l:\omega_l \in \boldsymbol{\omega}^{\text{CAND}}} \mu_l$
32:         if $\mu_l \leq 0$ then return $\hat{\boldsymbol{\omega}}^*$
33:         $\hat{\boldsymbol{\omega}}^* \leftarrow \hat{\boldsymbol{\omega}}^* \cup \{\omega_l\}$
34:         $\boldsymbol{\omega}^{\text{CAND}} \leftarrow \boldsymbol{\omega}^{\text{CAND}} \setminus \{\omega_l\}$
35:     return $\hat{\boldsymbol{\omega}}^*$

36: function SELECT($\boldsymbol{\omega}^{\text{ACC}}, \boldsymbol{\omega}^{\text{CAND}}, \hat{\boldsymbol{\omega}}^*$)
37:     $\hat{\gamma}^* \leftarrow \sum_{l:\omega_l \in \hat{\boldsymbol{\omega}}^*} \mu_l$
38:     for $\omega_l \in \boldsymbol{\omega}^{\text{CAND}}$ do
39:         if $\omega_l \in \hat{\boldsymbol{\omega}}^*$ then
40:             $\hat{\boldsymbol{\omega}}_l^* \leftarrow$ ASSEMBLE($\boldsymbol{\mu}, \boldsymbol{\omega}^{\text{ACC}}, \boldsymbol{\omega}^{\text{CAND}} \setminus \{\omega_l\}$)
41:         else
42:             $\hat{\boldsymbol{\omega}}_l^* \leftarrow$ ASSEMBLE($\boldsymbol{\mu}, \boldsymbol{\omega}^{\text{ACC}} \cup \omega_l, \boldsymbol{\omega}^{\text{CAND}} \setminus \{\omega_l\}$)
43:         $\hat{\gamma}_l^* \leftarrow \sum_{l:\omega_l \in \hat{\boldsymbol{\omega}}_l^*} \mu_l$
44:         $\psi_l \leftarrow \hat{\gamma}^* - \hat{\gamma}_l^*$
45:     return $\arg\max_l \psi_l$
```

Algorithm 1 shows how we apply the CSAR algorithm to the problem of optimizing communication under bandwidth constraints. The function OPTIMIZECOMMUNICATION (Lines 1–13) takes in the ego-agent's current beliefs about the transition function, $b_i(x)$, and its teammates, $b_i(m)$, and decides which observations from the set $\boldsymbol{\omega}$ to communicate. The resulting message must satisfy the ego-agent's bandwidth constraint β_i.

The function has a fixed computational budget B (in terms of iterations of the main loop) that it uses to evaluate observations and decide what to communicate. The CSAR algorithm dictates how much of this computational budget should be allocated to each of the K accept-reject decisions (Lines 14–18, see Chen et al. 2014 for derivation). The algorithm uses this budget allocation B_k to refine its reward distribution estimates of the candidate arms (Line 7).

EstimateRewardDistributions (Lines 19–27) is responsible for this task. The function carries out B_k forward simulations on each of the candidate observations. We begin each of these forward simulations by sampling a set of agent models (Line 21) and a transition function from the ego-agent's beliefs (Lines 22–23). The function EvaluateCommunication returns the reward obtained from communicating a candidate observation given that set of agent models and transition function (see Sect. 3.2). This reward is used to update the statistics associated with the observation (Lines 26–27).

Once the reward distributions have been updated, we assemble the estimated optimal message $\hat{\boldsymbol{\omega}}^*$ according to the current rewards and the bandwidth constraint (Line 8). The function Assemble (Lines 28–35) is responsible for this task. It starts by adding all of the previously accepted observations into the message. It then repeatedly adds the best remaining observation to the message until the bandwidth constraint is reached or the candidate pool is exhausted.

We use the optimal message estimate $\hat{\boldsymbol{\omega}}^*$ to help us select the observation we will decide on at this iteration (Lines 36–45). This message estimate corresponds to a particular accept-reject decision for each of the remaining candidate elements. We then assemble optimal message estimates $\hat{\boldsymbol{\omega}}_l^*$ in which we force the alternative decision to be made for each element ω_l (Lines 38–42). The difference in the reward expected to result from $\hat{\boldsymbol{\omega}}^*$ (the optimal message estimate) and $\hat{\boldsymbol{\omega}}_l^*$ (the optimal message estimate with the alternative decision about ω_l) is the suboptimality gap associated with ω_l (Line 44). The size of this suboptimality gap is a measure of the confidence of the associated reward distribution. We therefore select the observation with the largest suboptimality gap and decide whether to accept or reject it (Lines 10–12).

The running time of Algorithm 1 depends on the free parameter B as well as parameters given by the problem instance. The function OptimizeCommunication carries out B evaluations to refine its estimates of the reward distributions associated with each candidate observation. In each of these evaluations, the reward distribution for each candidate observation is updated. At most K such updates occur, since K is the number of initial candidate observations, and the pool of candidates shrinks as each accept-reject decision is made. Updating the reward distribution for a candidate observation requires a call to the function EvaluateCommunication, which carries out forward simulations for all $n - 1$ teammate agents (see Sect. 3.2). Forward simulating to time horizon T requires a policy computation at each time step, the cost of which depends on the particular planning algorithm used. An implementation with a naive shortest path planner would have complexity $O(|S| + |S||A|)$ (i.e. the sum of the number of vertices and edges in the state-action graph), but an incremental replanning algorithm requires this computation only at the first time step and then can quickly update the policy thereafter (Koenig and Likhachev 2005). Altogether, the worst-case complexity of OptimizeCommunication is $O(BKnT|S||A|)$, but careful implementation can reduce this significantly.

One drawback of the CSAR algorithm is that it assumes independence between the reward distributions of arms. Such an assumption is necessary for CSAR's tractability, but it may harm performance in tasks where the reward distributions are not independent. For example, there may be scenarios in which communicating the occupancy of one cell is only useful when combined with communicating the occupancy of a second cell. We leave study of this drawback as future work.

3.2 Evaluating communication actions

Evaluating candidate communications in terms of their expected effect on reward is typically expensive. In this section, we provide our method for performing this evaluation efficiently and review the key aspects of our formulation that enable this approach.

General multi-agent models are unfactored, assuming that team reward is a function of joint actions. Such models require planning in the joint action space, which scales exponentially with the number of agents. In contrast, we consider only factored multi-agent problems, where agents plan independently of one another. The computational cost of forward simulating such problems then scales linearly with the number of agents.

Furthermore, we assume that the transition function is deterministic. We can transform such a deterministic transition function into a directed graph where nodes are states and edges are actions. An agent can compute a policy by finding a shortest path through the state-action graph. We can further leverage incremental shortest-path planners to recompute the policy quickly at each step of the forward simulation (Koenig and Likhachev 2005).

Algorithm 2 specifies our method for evaluating candidate communications. The goal of this algorithm is to estimate

Algorithm 2 Evaluating Communication Actions

```
1: function EvaluateCommunication($\boldsymbol{\omega}_c$, $\tilde{\boldsymbol{m}}$, $\tilde{P}$)
2:     for $\tilde{m}_j \in \tilde{\boldsymbol{m}}$ do
3:         $(s_j, s_j^{\text{GOAL}}, b_j(x)) \leftarrow \tilde{m}_j$
4:         $\gamma_{A_j}^{\text{NoCom}} \leftarrow$ ForwardSimulate($\tilde{m}_j$, $\tilde{P}$)
5:         $b'_j(x) \leftarrow$ IncorporateObservations($b_j(x)$, $\boldsymbol{\omega}_c$)
6:         $\tilde{m}'_j \leftarrow (s_j, s_j^{\text{GOAL}}, b'(x))$
7:         $\gamma_{A_j}^{\text{COM}} \leftarrow$ ForwardSimulate($\tilde{m}'_j$, $\tilde{P}$)
8:     $\delta \leftarrow R_C(c) + \sum_j (\gamma_{A_j}^{\text{COM}} - \gamma_{A_j}^{\text{NoCom}})$
9:     return $\delta$

10: function ForwardSimulate($\tilde{m}_j$, $\tilde{P}$)
11:     $(s_j, s_j^{\text{GOAL}}, b_j(x)) \leftarrow \tilde{m}_j$
12:     $\gamma_A \leftarrow 0$
13:     for $t, \ldots, T$ do
14:         $\boldsymbol{\omega} \leftarrow$ Observe($s_j$, $\tilde{P}$)
15:         $b_j(x) \leftarrow$ IncorporateObservations($b_j(x)$, $\boldsymbol{\omega}$)
16:         $\pi \leftarrow$ ComputePolicy($s_j$, $s_j^{\text{GOAL}}$, $b_j(x)$)
17:         $a \leftarrow \pi(s_j)$
18:         $s_j \leftarrow \arg\max_{s'} \tilde{P}(\cdot|s_j, a)$
19:         $\gamma_A \leftarrow \gamma_A + R_A(s_j, a)$
20:     return $\gamma_A$
```

the effect a proposed communication will have on team reward given sampled agent models and a sampled transition function (see Algorithm 1). To measure this effect, we first forward simulate the agent models to get the baseline reward that occurs without any communication (Line 4). Next, we incorporate the communicated observation into the agent models' beliefs and repeat the forward simulation (Lines 5–7). The difference in reward is an estimate of the value of the communication (Line 8).

The function ForwardSimulate estimates the reward earned by an agent model $\tilde{m}_j$ in a world given by transition function $\tilde{P}$. At each time step of the forward simulation, the modeled agent first observes its environment and incorporates that observation into its belief (Lines 14–15). The modeled agent then plans and executes an action and arrives in an updated state (Lines 16–18). The function returns the cumulative action reward earned by the modeled agent (Line 20).

The function EvaluateCommunication of Algorithm 2 is called from within the function EstimateRewardDistributions of Algorithm 1, where it is used to evaluate the effect of communicating a candidate observation and thus update the estimated reward distribution associated with that observation.

3.3 Making sequential communication decisions with OCBC

In Algorithm 3, we show how we use OCBC within a sequential decision-making agent. This algorithm provides context

Algorithm 3 Sequential Decision Algorithm

```
   For convenience, let $s_i(\cdot)$, $s_i^{\text{GOAL}}$, $a_i(\cdot)$, $\boldsymbol{\omega}_i(\cdot)$, $c_i(\cdot)$ be global variables
   shared among all functions
1: function MakeSequentialDecisions( )
2:     $b_i(x), b_i(m) \leftarrow$ InitializeBeliefs( )
3:     for $t \in \{1, \ldots, T\}$ do
4:         $\boldsymbol{\omega}_i(t), \boldsymbol{c}(t-1) \leftarrow$Sense( )
5:         $b_i(x), b_i(m) \leftarrow$ UpdateBeliefs($b_i(x)$, $b_i(m)$)
6:         $\pi, c_i(t) \leftarrow$ Plan($b_i(x)$, $b_i(m)$)
7:         Act($\pi$)
8:     $\gamma_i \leftarrow \sum_{t=1}^{T} R_A(s_i(t), a_i(t)) + R_C(c_i(t))$
9:     return $\gamma_i$

10: function InitializeBeliefs( )
11:     for $s \in S$ do
12:         $b_i(x_s) \leftarrow p_s$
13:     for $j \in I \setminus \{i\}$ do
14:         $m_j \leftarrow (s_j, s_j^{\text{GOAL}}, b_j(x))$
15:         $b_i(m_j) \leftarrow \bigcup_{1,\ldots,M} \{m_j\}$
16:     $b_i(m) \leftarrow \bigcup_{j \in I \setminus \{i\}} b_i(m_j)$
17:     return $b_i(x), b_i(m)$

18: function Sense( )
19:     $\boldsymbol{\omega}_i(t) \leftarrow$ Observe($s_i(t)$, $P$)
20:     $\boldsymbol{c}(t-1) \leftarrow$ReceiveCommunications( )
21:     return $\boldsymbol{\omega}_i(t), \boldsymbol{c}(t-1)$

22: function UpdateBeliefs($b_i(x)$, $b_i(m)$)
23:     $\boldsymbol{\omega}_{\text{Update}} \leftarrow \boldsymbol{\omega}_i(t) \cup \boldsymbol{c}(t-1)$
24:     $b'_i(x) \leftarrow$ IncorporateObservations($b_i(x)$, $\boldsymbol{\omega}_{\text{Update}}$)
25:     $b'_i(m) \leftarrow$ UpdateTeamBeliefs($b_i(m)$, $\boldsymbol{\omega}_{\text{Update}}$)
26:     return $b'_i(x), b'_i(m)$

27: function Plan($b_i(x)$, $b_i(m)$)
28:     $\pi \leftarrow$ ComputePolicy($s_i(t)$, $s_i^{\text{GOAL}}$, $b_i(x)$)
29:     $\boldsymbol{\omega}_i^{\text{ALL}} \leftarrow \bigcup_{\tau \in \{1,\ldots,t\}} \boldsymbol{\omega}_i(\tau)$
30:     $c_i(t) \leftarrow$ OptimizeCommunication($b_i(x)$, $b_i(m)$, $\boldsymbol{\omega}_i^{\text{ALL}}$)
31:     return $\pi, c_i(t)$

32: function Act($\pi$)
33:     $a_i(t) \leftarrow \pi(s_i(t))$
34:     $s_i(t+1) \leftarrow \arg\max_{s'} P(\cdot|s_i(t), a_i(t))$
35:     Communicate($c_i(t)$)
```

for using OCBC as well as a means for evaluating its performance (see Sect. 4).

3.3.1 Initialization

The algorithm begins by initializing the ego-agent's beliefs via the function InitializeBeliefs (Lines 10–17). The agent starts with some prior p_s for each transition function value $b_i(x_s)$ (Line 12). The ego-agent also initializes its beliefs about each of its teammates (Lines 13–16). In our evaluation, we assume a strong prior on this belief, that is, we assume that the ego-agent knows the initial state and transition belief of each of its teammates. Such a prior could come from an offline sharing step before the task begins. We imple-

Algorithm 4 Incorporating Observations into a Transition Belief

```
1: function IncorporateObservations($b(x)$, $\boldsymbol{\omega}$)
2:     for $\omega^{(s)} \in \boldsymbol{\omega}$ do
3:         $b'(x_s) \leftarrow \eta O(\omega^{(s)}|x_s) b(x_s)$
4:     return $b'(x)$
```

ment the belief over agent models as a set of particles (see Sect. 3.4), so the initial particle set contains M copies of the agent model m_j, where M is a user-specified parameter.

3.3.2 Sensing

At each time step t, the ego-agent first senses its environment (Line 4), making observations $\boldsymbol{\omega}_i(t)$ drawn from the observation function $O(\omega^{(s)}|x_s)$ (Line 9). Additionally, the ego-agent receives the communications $\boldsymbol{c}(t-1)$ transmitted by its teammates at the previous time step (Line 20).

3.3.3 Updating beliefs

Next, the agent updates its beliefs based on the result of the sensing step (Line 5). The ego-agent aggregates all the observations it made directly or received via communication (Line 23). It incorporates these observations into its transition belief via the function IncorporateObservations (see Algorithm 4). The ego-agent then updates its beliefs about its teammates via the function UpdateTeamBeliefs (see Sect. 3.4 and Algorithm 5).

3.3.4 Planning

The ego-agent computes a policy π based on its current state $s_i(t)$ and its transition belief $b_i(x)$ (Line 28). Then, it selects a (possibly empty) set of observations to communicate using the function OptimizeCommunication, the implementation of which is our primary contribution (see Algorithm 1).

3.3.5 Acting

Finally, the agent executes action $a_i(t)$ based on the policy π and arrives in state $s_i(t+1)$ (Lines 33–34). The agent then broadcasts its selected set of observations to all its teammates (Line 35).

3.4 Updating agent beliefs

3.4.1 Transition beliefs

Algorithm 4 shows how we update an agent's transition belief based on a set of observations. Recall that we assume

Algorithm 5 Updating Team Beliefs

```
1: function UpdateTeamBeliefs($b_i(m)$, $\boldsymbol{\omega}_{\text{UPDATE}}$)
2:     for $b_i(m_j) \in b_i(m)$ do
3:         for $m^{(k)} \in b_i(m_j)$ do
4:             $\left(s, s^{\text{GOAL}}, b(x)\right) \leftarrow m^{(k)}$
5:             $\pi \leftarrow$ ComputePolicy($s$, $s^{\text{GOAL}}$, $b(x)$)
6:             $a \leftarrow \pi(s)$
7:             Sample $\tilde{x} \sim b(x)$
8:             $\tilde{P} \leftarrow$ ConstructTransitionFunction($\tilde{x}$)
9:             $s' \leftarrow \arg\max_{s'} \tilde{P}(\cdot|s, a)$
10:            $\boldsymbol{\omega} \leftarrow$ Observe($s'$, $\tilde{P}$)
11:            $b'(x) \leftarrow$ IncorporateObservations($b(x)$, $\boldsymbol{\omega}$)
12:            $m'_k \leftarrow (s', s^{\text{GOAL}}, b'(x))$
13:            $w_k \leftarrow \prod_{\omega^{(s)} \in \boldsymbol{\omega}_{\text{UPDATE}}} \mathbb{P}\left(\omega^{(s)}|b'(x_{s'})\right)$
14:        $b'_i(m_j) \leftarrow \bigcup_{1,\dots,M}$ Sample $m'_k$ with probability $\propto w_k$
15:    return $b'_i(m)$
```

the beliefs about transition elements are independent of one another. Therefore, observation $\omega^{(s)}$ only affects belief element $b(x_s)$. We use Bayes rule to compute the new value of the belief for each affected element (Line 3).

3.4.2 Teammate beliefs

Algorithm 5 details our method of updating the ego-agent's beliefs about its teammate models. The algorithm takes in the existing beliefs $b_i(m)$ as well as a set of new observations $\boldsymbol{\omega}_{\text{UPDATE}}$. It performs a particle filter update on the belief $b_i(m_j)$ about each teammate j (Lines 3–14).

First, each particle $m^{(k)}$ is evolved forward one step (Lines 4–12). Recall that each particle $m^{(k)}$ is an agent model that contains a state, goal state, and transition belief. We then plan and execute an action for the agent model (Lines 5–6). To simulate the next time step for the agent model, we sample from the model's transition element beliefs and construct a transition function $\tilde{P}$ (Lines 7–8). We use $\tilde{P}$ to find the new state s' and generate an observation $\boldsymbol{\omega}$ (Lines 9–10). We then incorporate this observation into the particle's transition belief to get evolved belief $b'(x)$ (Line 11). At this point, we assemble s', $s^{\text{GOAL}}_{m_k}$, and $b'(x)$ into an evolved particle m'_k. We assign the particle weight w_k based on the likelihood of the actual observations $\boldsymbol{\omega}_{\text{UPDATE}}$ given the updated model (Line 13). We then re-sample the particles according to these weights to get the updated set $b'_i(m_j)$ (Line 14).

4 Evaluation

In this section, we evaluate our method on the multi-robot navigation task introduced in Sect. 2.1. We structure our evaluation around the three communication paradigms introduced in Sect. 2.2. As we have presented it so far, OCBC is a part of the *Fixed-Bandwidth* paradigm. The closest existing work (Unhelkar and Shah 2016) is in the *Fixed-Cost* paradigm. We therefore evaluate variants of OCBC for the *Fixed-Cost* and *Proportional-Cost* paradigms that allow us to compare it to this existing method.

We first detail the experimental setup (Sect. 4.1). Then, we compare OCBC to ConTaCT (Unhelkar and Shah 2016) in the *Fixed-Cost* and *Proportional-Cost* paradigms (Sect. 4.2). Finally, we compare OCBC to a randomized baseline method in the *Fixed-Bandwidth* paradigm (Sect. 4.3).

4.1 Experimental setup

In the experiments of Sect. 4.2, we use five agents in each simulation. In Sect. 4.3, we vary the number of agents between two and ten.

We begin each simulated trial by generating a 10×10 gridmap. The occupancy of each grid cell is determined by a Bernoulli trial with probability 0.3. Such a randomized method can yield maps without feasible paths between certain cells. We therefore test that the resulting gridmap is fully connected; that is, any given cell in the map must be reachable from every other cell. If the gridmap is not fully connected, we replace it with a new randomly generated map until the condition is met.

Once the map is generated, we select a shared goal cell and agent start cells uniformly at random from among the unoccupied cells. Because of the random nature of this process, some generated problem instances are not interesting tests of communication. For example, agents that begin on opposite sides of a map may not benefit at all from sharing observations with one another. We can detect such cases by conducting two initial simulations: in the first, agents automatically share all observations, and in the second, agents share nothing. If both simulations yield the same result, we reject the trial as uninteresting and generate a new gridmap, a new goal location, and new agent locations.

Each simulated trial returns the reward, $\gamma(\chi) = \gamma_A(\chi) + \gamma_C(\chi)$, earned by the team using method χ. We are interested in measuring the effect of communication on task performance. In this case, task performance is measured by action reward, which is given by $\gamma_A(\chi)$. To measure the effect of communication on task performance, we repeat the trial without any inter-robot communication to obtain the value $\gamma(\textsc{NoCom}) = \gamma_A(\textsc{NoCom}) + \gamma_C(\textsc{NoCom})$. We then compute the difference in action reward earned during the two trials, that is,

$$\delta(\chi) = \gamma_A(\chi) - \gamma_A(\textsc{NoCom}). \tag{2}$$

We normalize this value by dividing by the corresponding value for the method AllCom, in which agents automatically share all observations. This normalized value is given by

$$\hat{\delta}(\chi) = \frac{\delta(\chi)}{\delta(\textsc{AllCom})}. \tag{3}$$

In other words, $\hat{\delta}(\chi)$ measures how much the communication of method χ helps task performance relative to the full communication baseline method.

For each method, we run the same set of 1000 trials. Each data point represents the mean of $\hat{\delta}(\chi)$ for these trials, and the error bars represent one standard error on the mean. In our experiments, we set OCBC's computational budget at $B = 1000$ and maintain beliefs consisting of $M = 50$ particles.

4.2 Fixed-cost and proportional-cost communication

To compare OCBC to existing work (Unhelkar and Shah 2016), we present two variants *OCBC, Fixed-Cost* and *OCBC, Proportional-Cost*.

In *OCBC, Proportional-Cost*, the ego-agent has no limit on its available bandwidth (i.e. $\beta_i \to \infty$), but each communicated observation incurs some cost (i.e. $R_C(\cdot) > 0$). This causes the function Assemble of Algorithm 1 to return all observations with positive expected reward regardless of how large the resulting message is. Although *OCBC, Proportional-Cost* is a variant of OCBC, it still includes both of our primary contributions (i.e. optimizing message content and evaluating messages with forward simulation).

By contrast, *OCBC, Fixed-Cost* only includes the latter contribution (i.e. evaluating messages with forward simulation). It uses the same sampling method as OCBC, but only evaluates messages containing all available observations.

We compare the performance of these methods to that of *ConTaCT*, a method introduced by Unhelkar and Shah (2016). Our method has several key distinctions from *ConTaCT*. First, the *ConTaCT* ego-agent maintains only a single estimate of the transition function and of its teammates, whereas OCBC maintains belief distributions over these values (see Sect. 3.4). Second, *ConTaCT* estimates the no-communication reward by computing the expected reward of an agent's previously declared policy in the ego-agent's updated transition function estimate. This fixed-policy evaluation does not consider the future observations and associated re-planning of other agents, which OCBC models through forward simulation (see Sect. 3.2). Finally, *ConTaCT* reasons only about *when* to communicate, whereas OCBC also considers *what* to communicate, which allows OCBC to operate under bandwidth constraints (see Sect. 3.1).

ConTaCT belongs to the *Fixed-Cost* paradigm and thus reasons only about *when* to communicate. The primary distinction between *OCBC, Fixed-Cost* and *ConTaCT* is our forward simulation method. Unlike *ConTaCT*, our method considers the future observations and re-planning of other agents as well as the uncertainty in the ego-agent's beliefs.

To compare the cost-based methods, we vary the modeled cost of communication in the range $0.01 \leq R_C(\cdot) \leq 20$ and measure the resulting task performance and proportion of observations communicated. Figure 3 shows the results of this experiment with the measured number of communications on the horizontal axis and the measured task performance on the vertical axis. Note that the controlled parameter in the experiment is the communication cost, and both the number of communications and task performance are measured values. When the modeled cost of communication is high, all methods communicate little, leading to low task performance. As the modeled cost parameter decreases, the amount of communication and task performance both increase.

Our partial method, *OCBC, Fixed-Cost*, performs similarly to *ConTaCT* at low levels of communication, but significantly outperforms *ConTaCT* as the amount of communication increases. At the top right portion of the figure, *OCBC, Fixed-Cost* communicates less than 10% of its observations while achieving approximately 85% of the task performance of the method that communicates all observations.

Our full method, *OCBC, Proportional-Cost*, outperforms *ConTaCT* across the entire tested range of communication levels. For a given level of communication, *OCBC, Proportional-Cost* achieves significantly higher task performance compared to *ConTaCT*. Furthermore, to achieve the same level of task performance, our method uses more than an order of magnitude fewer communications. At the extreme end of its range, *OCBC, Proportional-Cost* achieves approximately 55% of the task performance of the full communication method with less than 1% of the communications.

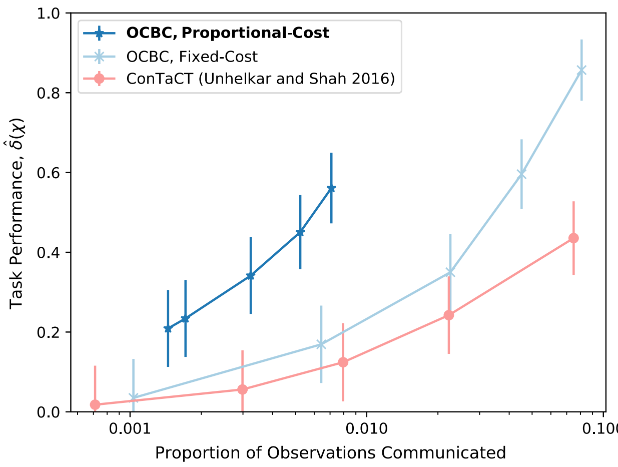


Fig. 3 Effect of communication on task performance as a function of the proportion of observations communicated. To achieve the different levels of communication, we vary the modeled cost of communication within each method. As the cost of communication decreases, the methods communicate more often and thus achieve higher task performance. Compared to previous work (*ConTaCT*, Unhelkar and Shah 2016), our full method (*OCBC, Proportional-Cost*) yields higher task performance for a given level of communication. Furthermore, to achieve the same level of task performance, our full method communicates more than an order of magnitude less

4.3 Fixed-bandwidth communication

We now evaluate OCBC's performance in the *Fixed-Bandwidth* paradigm. We fix the bandwidth available to the robots, β, and vary the number of robots sharing that bandwidth. The robots use a round-robin scheduling method, with each robot i having bandwidth allocation $\beta_i(t)$ at time step t. As the number of agents sharing the bandwidth increases, the bandwidth allocation per agent decreases. We evaluate OCBC against a baseline method that randomly selects observations.

Figure 4 shows the effect of communication on task performance for variable-size robot teams. The total bandwidth available in all experiments is $\beta = 4$, meaning that up to 4 observations can be communicated across the entire team at each time step.

We first point out the general trend present in the methods: as more robots share the same amount of bandwidth, task performance decreases. This may be surprising at first since the same number of observations are still transmitted at each time step. However, the value of communications varies, and the round-robin bandwidth allocation cannot account for this. As a part of a large team sharing a small amount of bandwidth, a robot that makes a particularly interesting observation may

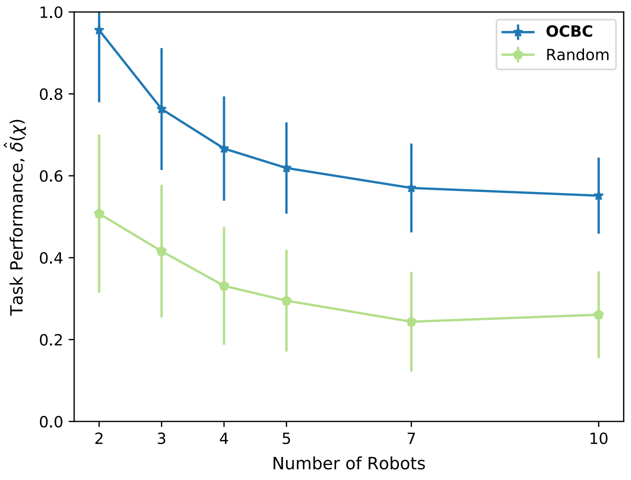


Fig. 4 Effect of communication on task performance as a function of the number of simulated robots sharing a fixed amount of bandwidth. As team size increases, each robot has less opportunities to communicate, leading to decreased performance. Our method (*OCBC*), achieves significantly better task performance compared to a randomized baseline method

have to wait several time steps for its turn to share that observation.

We observe that across the range of team sizes, our method significantly outperforms the *Random* baseline method. Furthermore, our method achieves normalized task performance close to 1 in the 2-robot experiment. This means that, in that experiment, our method has similar task performance to the baseline method sharing all observations without a bandwidth constraint. Our method achieves this performance while communicating at least 50% fewer observations.

We also point out that this experiment features simulations with up to 10 agents in an environment with 100 states and 4 available actions. This is one of the largest-scale experiments to date in the communication decision-making literature, demonstrating the scalability of our method.

5 Related work

5.1 Communicating based on decision theory

Decision-theoretic approaches like OCBC evaluate communication actions based on their expected effect on reward. Computing this value directly for general agent models like Dec-MDPs or Dec-POMDPs is prohibitively complex (Pynadath and Tambe 2002; Goldman and Zilberstein 2004). To avoid direct computation and its associated costs, Williamson et al. (2008, 2009) measure the information content in recent messages and use this as a proxy for a message's value. More concretely, their method computes the KL divergence between an agent's current belief and its belief at the time of its last communication. If this divergence is sufficiently large compared to some threshold parameter, the agent decides to communicate.

Carlin and Zilberstein (2009a, b) and Becker et al. (2009) examine the effect of common myopic assumptions made by existing methods. Specifically, they analyze the assumptions that future communication will not occur and that other agents will not initiate communication themselves. They argue that these assumptions lead to over-communication and present an algorithm free from these assumptions. However, the complexity of the resulting algorithm limits it to very small problems. OCBC makes these assumptions for the sake of tractability.

Unhelkar and Shah (2016) propose ConTaCT, which deliberately deemphasizes uncertainty to make communication decisions tractable in larger domains. In ConTaCT, the ego-agent maintains a belief about the world as well as an estimate of other agents' beliefs. By evaluating the expected result of sharing its available observations, the ego-agent decides whether or not to communicate. We discussed ConTaCT further in Sect. 4 and evaluated its performance against that of OCBC.

5.2 Selecting what to communicate

Because of the complexities already involved in deciding *when* to communicate, few papers have addressed the question of *what* to communicate. Roth et al. (2006) and Roth (2007) point out this deficit and provide one of the few methods for optimizing the content of communication messages. Specifically, they use greedy hill-climbing optimization to select observations to include in a communication. Their algorithm is built on top of their previous coordination method in which all agents act on common knowledge (Roth et al. 2005). They then try to select the set of observations to add to this common knowledge to maximize team reward. Because they use a very general multi-agent model, their approach does not scale beyond small problem domains.

Giamou et al. (2018) take a task-oriented approach to selecting what to communicate, proposing a communication planning framework for cooperative SLAM. Specifically, they find a solution for exchanging the minimal amount of raw sensory data without missing out on potential loop closures. While our approach is also concerned with selecting observations to share, their method does not extend beyond the specific problem of cooperative SLAM.

5.3 Communicating to maintain coordination

In many multi-agent problems, agents must tightly coordinate their actions. Such coordination typically requires significant amounts of inter-agent communication. Therefore, some methods reason about when communication is necessary to maintain coordination.

In Roth et al. (2005) and Roth (2007), agents generate an offline centralized policy that maps possible joint beliefs to joint actions. During execution, agents use this policy to select actions based on the current joint belief. When agents make observations, they consider how communicating those observations would change the joint belief and how the updated joint belief would affect team performance.

Wu et al. (2011) also maintain coordination by having agents act only on common knowledge. Agents communicate whenever they detect an inconsistency in their shared belief. This guarantees that agents will remain coordinated even when they forgo communication for some time.

Best et al. (2018b) present a communication planning algorithm suitable for the Dec-MCTS coordination framework (Best et al. 2018a). In Dec-MCTS, agents maintain belief distributions over the possible future action sequences of their teammates. Best et al. (2018b) reason about when an agent should request information from a teammate to update this distribution. More specifically, when a belief becomes sufficiently uncertain, the agent requests an updated distribution from the teammate.

In this paper, we focus on factored multi-agent problems where coordination is not a concern. Rather, agents communicate observations to help their teammates perform their respective independent tasks.

5.4 Deep reinforcement learning

Recent papers (Foerster et al. 2016; Sukhbaatar et al. 2016) have used deep reinforcement learning to learn communication policies. Such methods do not have a defined language of symbols for the agents to transmit. Rather, the agents learn to use a continuous-valued broadcast communication channel. Foerster et al. (2016) use such a method to solve communication riddles and multi-agent computer vision problems. Sukhbaatar et al. (2016) show results for multi-turn games and communication at a traffic junction. Such data-driven methods show promise in the demonstrated domains, but the continuous-valued broadcast channel limits the types of information that can be communicated by such systems.

6 Conclusion

In this paper, we proposed OCBC, an approach to Optimizing Communication under Bandwidth Constraints. OCBC uses forward simulation to evaluate possible communication actions and incorporates those evaluations into a bandit-based combinatorial optimization algorithm that computes an approximately optimal set of observations to communicate. OCBC is designed for collaborative multi-agent problems with a deterministic but unknown transition function, which includes tasks where multiple robots operate in previously unexplored terrain. We showed that OCBC outperforms its closest existing competitor at a simulated multi-robot navigation task, achieving higher task performance while communicating more than an order of magnitude less.

Acknowledgements This material is based upon work supported by the National Science Foundation (NSF) Graduate Research Fellowship Program under Grant No. DGE 1256260, as well as NSF Grant Nos. CCF 1442773 and NRI 1830615. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

Becker, R., Carlin, A., Lesser, V., & Zilberstein, S. (2009). Analyzing myopic approaches for multi-agent communication. *Computational Intelligence*, *25*(1), 31–50.

Becker, R., Zilberstein, S., Lesser, V., & Goldman, C. V. (2004). Solving transition independent decentralized Markov decision processes. *Journal of Artificial Intelligence Research*, *22*, 423–455.

Best, G., Cliff, O. M., Patten, T., Mettu, R. R., & Fitch, R. (2018a). Dec-MCTS: Decentralized planning for multi-robot active perception. *International Journal of Robotics Research*, *38*(2–3), 316–337.

Best, G., Forrai, M., Mettu, R., & Fitch, R. (2018b). Planning-aware communication for decentralised multi-robot coordination. In *Proceedings of the IEEE international conference on robotics and automation*.

Bianchi, G. (1998). IEEE 802.11-saturation throughput analysis. *IEEE Communications Letters*, *2*(12), 318–320.

Carlin, A., & Zilberstein, S. (2009a). Myopic and non-myopic communication under partial observability. In *Proceedings of the IEEE/WIC/ACM international joint conference on web intelligence and intelligent agent technology* (pp. 331–338). IEEE.

Carlin, A., & Zilberstein, S. (2009b). Value of communication in decentralized POMDPs. In *Proceedings of the AAMAS workshop on multi-agent sequential decision making in uncertain domains* (pp. 16–21).

Chen, S., Lin, T., King, I., Lyu, M. R., & Chen, W. (2014). Combinatorial pure exploration of multi-armed bandits. In *Proceedings of the advances in neural information processing systems conference* (pp. 379–387).

Foerster, J., Assael, Y., de Freitas, N., & Whiteson, S. (2016). Learning to communicate with deep multi-agent reinforcement learning. In *Proceedings of the advances in neural information processing systems conference* (pp. 2137–2145).

Giamou, M., Khosoussi, K., & How, J. P. (2018). Talk resource-efficiently to me: Optimal communication planning for distributed slam front-ends. In *Proceedings of the IEEE international conference on robotics and automation*.

Goldman, C., & Zilberstein, S. (2004). Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of Artificial Intelligence Research*, *22*, 143–174.

Hiertz, G. R., Max, S., Rui, Z., Denteneer, D., & Berlemann, L. (2007). Principles of IEEE 802.11s. In *Proceedings of the international conference on computer communications and networks*.

Koenig, S., & Likhachev, M. (2005). Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics*, *21*(3), 354–363.

Pynadath, D. V., & Tambe, M. (2002). The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, *16*, 389–423.

Roth, M. (2007). Execution-time communication decisions for coordination of multi-agent teams. PhD thesis, Carnegie Mellon University.

Roth, M., Simmons, R., & Veloso, M. (2005). Reasoning about joint beliefs for execution-time communication decisions. In *Proceedings of the international conference on autonomous agents and multi-agent systems*.

Roth, M., Simmons, R., & Veloso, M. (2006). What to communicate? execution-time decision in multi-agent pomdps. In *Distributed autonomous robotic systems 7* (pp. 177–186). Tokyo: Springer

Shrader, B., & Ephremides, A. (2007). Random access broadcast: Stability and throughput analysis. *IEEE Transactions on Information Theory*, *53*(8), 2915–2921.

Sukhbaatar, S., Szlam, A., & Fergus, R. (2016). Learning multiagent communication with backpropagation. In *Proceedings of the advances in neural information processing systems conference* (pp. 2244–2252).

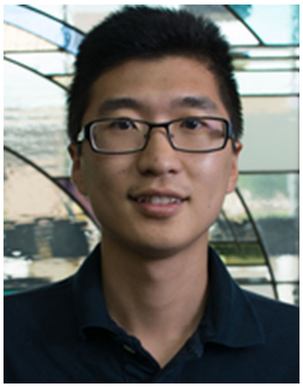
Unhelkar, V., & Shah, J. (2016). ConTaCT: Deciding to communicate during time-critical collaborative tasks in unknown, deterministic domains. In *Proceedings of the AAAI national conference on artificial intelligence* (pp. 2544–2550).

Williamson, S., Gerding, E., & Jennings, N. (2008). A principled information valuation for communications during multi-agent coordination. In *Proceedings of the AAMAS workshop on multi-agent sequential decision making in uncertain domains*.

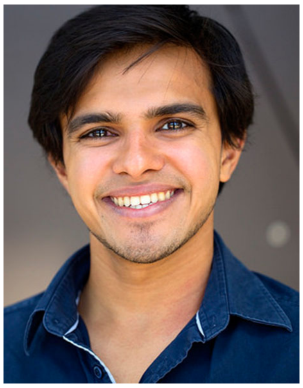
Williamson, S., Gerding, E., & Jennings, N. (2009). Reward shaping for valuing communications during multi-agent coordination. In *Proceedings of the international conference on autonomous agents and multi-agent systems*.

Wu, F., Zilberstein, S., & Chen, X. (2011). Online planning for multi-agent systems with bounded communication. *Artificial Intelligence*, *175*(2), 487–511.

Ryan J. Marcotte is a Ph.D. Candidate in Computer Science and Engineering at the University of Michigan. He received the B.S. degree (Summa Cum Laude) in Electrical Engineering from the University of Texas at Dallas in 2015 and the M.S. degree in Computer Science and Engineering from the University of Michigan in 2016. His research interests include communication and planning for multi-robot systems.



Xipeng Wang is a Ph.D. candidate in Computer Science Engineering at the University of Michigan. His research interests lie in theoretical and experimental large-scale mapping and localization for ground vehicles. Before coming to UM, he received his B.E. from Xi'an Jiao Tong University, China.



Dhanvin Mehta is a Ph.D. Candidate in Computer Science and Engineering at the University of Michigan. He received the B.Tech. (2014) and M.Tech. (2015) degrees in Computer Science and Engineering from the Indian Institute of Technology, Madras. His research enables reliable autonomous navigation in dynamic social environments.

Edwin Olson is an Associate Professor of Computer Science and Engineering at the University of Michigan. He is the director of the APRIL robotics lab, which studies Autonomy, Perception, Robotics, Interfaces, and Learning. His active research projects include mapping methane in landfills, multi-robot search and rescue, communication, railway safety, and automobile autonomy and safety. He received a Ph.D. from the Massachusetts Institute of Technology in 2008 for his work in robust robot mapping.