Autonomous Robots manuscript No.
(will be inserted by the editor)

Cooperative Queuing Policies for Effective Scheduling of Operator Intervention

Masoume M. Raeissi · Alessandro Farinelli

Received: date / Accepted: date

Abstract We consider multi-robot applications, where a team of robots can ask for the intervention of a human operator to handle difficult situations. As the number of requests grows, team members will have to wait for the operator attention, hence the operator becomes a bottleneck for the system. Our aim in this context is to make the robots learn cooperative strategies to decrease the idle time of the system by modeling the operator as a shared resource. In particular, we consider a **balking** queuing model where robots decide whether or not to join the queue and use multi-robot learning to estimate the best cooperative policy. In more detail, we formalize the problem as Decentralized Markov Decision Process and provide a suitable state representation, so to apply an independent learners approach. We evaluate the proposed method in a robotic water monitoring simulation and empirically show that our approach can significantly improve the team performance, while being computationally tractable.

1 Introduction

In many multi-robot scenarios, such as environmental monitoring [26] or search and rescue [27, 13], one or few operators are required to interact with a team of robots to perform complex tasks in challenging environments. Robots, specially at field sites, are often subject to unexpected events, that can not be managed without the intervention of operators. For example, in an environmental monitoring application, robots might face extreme environmental events (e.g., water currents) or moving obstacles (e.g., animal approaching the robots). In such scenarios, the operator often needs to interrupt the activities of individual team members to deal with particular situations.

University of Verona
Verona, Italy
E-mail: masoume.raeissi@univr.it University of Verona
Verona, Italy
E-mail: alessandro.farinelli@univr.it

The operator's monitoring and supervisory role in these scenarios becomes critical, particularly when the team size grows larger. To decrease the operator's monitoring task and give him/her more time to focus on robots that need attention, several approaches consider the concept of self-reflection [21], where robots are able to identify their potential issues and ask for the intervention of the operator by sending a request. However, large teams can easily overwhelm the operator with several requests, hence hindering the team performance. Consequently, team members have to wait for the operator's attention, and the operator becomes a bottleneck for the system. Queuing is a natural way to manage and address this problem. Previous research try to enhance the performance of the system (i.e., decreasing the time spent by robots waiting for the operator) considering various queue disciplines (e.g. First in First Out (FIFO) and Shortest Job First (SJF)) [15, 5] or prioritizing such requests [19]. In both cases, the queue size may grow indefinitely as no robot will leave the queue before receiving the operator's attention.

To deal with this problem, we focus on balking queue model [16], in which the users/agents (i.e robots requesting attention) can decide either to join the queue or balk. Such decisions are typically based on a threshold value, that is computed by assigning a generic reward for receiving the service and a cost for waiting in the queue to each agent. When applying this model to a robotic application, there is no clear indication how such thresholds can be computed. More important, this model does not consider the cost of balking (i.e. the cost of a potential failure that robots can suffer by trying to overcome difficult situations without human intervention). Considering this, our focus is to devise an approach that allows the team of robots to learn cooperative balking strategies to make better use of a shared queue. Therefore, we frame the above problem as a Decentralized

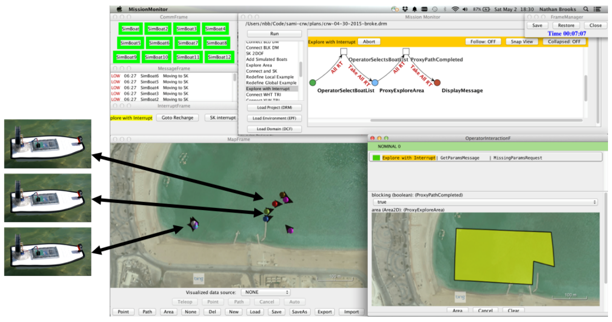


Fig. 1 Water monitoring simulation tool.

Markov Decision Process (Dec-MDP) in which, the team of robots must cooperate to optimize the team idle time. Finding optimal decentralized policies is often hard due to the partial observability and limited communications. Thus, our goal is to provide a scalable state representation by adding the state of the queue as an extra feature to the robots' local states and solve the underlying Dec-MDP problem using multiple independent learners [1]. We illustrate that, this additional feature will improve the team performance over our main evaluation metric (i.e. team idle time). In more detail, this paper provides the following contributions to the state of the art: (i) We model the human operator as a shared resource that robots can access using a balking queue; robots identify their needs and decide whether to join the queue (hence waiting to have access the resource) or not. (ii) We formalize the problem as Dec-MDP and provide a tractable state representation to learn the balking policies for each robot. (iii) Finally, we evaluate the performance of our model by comparing the team idle time to the state-of-the-art queue disciplines. Overall, the experimental results show that, the use of our model decreases the total idle time up to 68% over FIFO (without balking) and increases the team reward up to 56% comparing to the other learning models.

2 Background

In this section, first we review the state of the art in robotic studies, where robots ask for operator's attention. Afterwards, we present a brief introduction to the Balking Queue[16] model in which, users/agents decide to join the queue or not. Then, we present a brief review of Dec-MDP[1, 9] as the basis model for decision making under uncertainty in multi-robot scenarios.

2.1 Human-Multi-Robot Interaction

Human-multi robot interaction is an active field of research with many different research issues, including team plan representation [25, 14], multi-modal interaction [22, 10] and mixed initiative planning [2]. Here, we consider approaches, which focus on how to allocate operator attention to a set of robots. Many work in this area consider that, the robots can perceive their situations and inform or ask the operator for help. For example, the work by [20] proposes the idea of a single service robot asking for the help of humans.

Considering larger multi-robot teams with small number of operators, several robots may need the operator's attention at the same time. Hence, the requests must be queued for being processed later on. Authors in [5, 15] explore different queue disciplines to enhance the performance of the system. However, keeping robots idle until the operator becomes available might decrease the overall team efficiency. In contrast, we focus on a specific queuing model with balking property [16], where robots decide either to join the queue or not.

The concept of Adjustable Autonomy or mixed initiative has been the basis of many research in the field of human-multi-robot interaction. The key issue in this setting is to devise effective techniques to decide whether a transfer of control should occur and when this should happen. Different techniques have been proposed to address this challenge, for example, [12, 6, 24] consider that the robot will ask for human help/intervention when the expected utility of doing so is higher than performing the task autonomously, or when the uncertainty of the autonomous decision is high [11, 4] or when the autonomous decision can cause significant harm [7]. However, these decision making solutions usually have been considered as individual one-step decisions without considering the long-term cost or the consequences of decisions on other team members (if any). Plan-based approaches to mixed initiative systems [29, 3, 2] do consider long term costs of actions. However, in this work we focus on a specific interaction modality between the operator and the multi-robot system (i.e., a queuing system) and we aim at explicitly addressing uncertainty in action execution (i.e., probability of failures when acting autonomously). In more detail, our focus is on a specific queuing model with balking property [16] and on finding cooperative strategies where all robots learn concurrently to minimize the idle time of the system.

2.2 Balking Queue Model

The first mathematical model of a queuing system with rational users was formulated by Naor [16]. In his model, users upon their arrival decide according to a threshold value whether to join the queue or not (balk). The individual's optimizing

[1] Some part of this work appears in [18]. That work describes basic ideas and preliminary results, here we provide a more detailed description of the methodologies, and more extensive empirical analysis.

strategy is straightforward, a customer will join the queue while n other customers are already in the system if

$$R - n \cdot C\frac{1}{\mu} \geq 0 \tag{1}$$

where a uniform cost C for staying in the queue and a similar reward R for receiving service are assigned to each user and μ is the intensity parameter of exponentially distributed service time. In more detail, following a standard approach in literature the model proposed in [16] assumes that the time required to service a request follows an exponential distribution with parameter μ, hence the probability density function for the service time is given by $f(x;\mu) = \mu e^{-\mu x}$ where $x \geq 0$ and 0 otherwise.

Thus, $n = \lfloor \frac{R\mu}{C} \rfloor$ serves as a threshold value for balking, that is if the number of users waiting in the queue is greater than n, the newly arrived user will not join the queue. In a multi-robot application, this threshold and decision must be computed carefully. Our focus is on showing how the elements (i.e. reward and cost) of balking strategy should be adjusted according to a practical robotics scenario.

2.3 Decentralized Markov Decision Process (Dec-MDP)

The decision of whether to join the queue or not for each situation of each robot will impact the future decisions of the entire team. As a result, we are concerned here with team sequential decision making problems, in which the team's utility depends on a sequence of decisions.

A Dec-MDP is defined by a tuple $\langle S, A, P, R \rangle$ where: S is the set of world states which is factored into n components, $S = S_1 \times ... \times S_n$. In a special case (i.e. Factored n-agent Dec-MDP), S_i refers to the local state of agent i. In Dec-MDP, the state is jointly fully observable which means that the aggregated observations made by all agents determines the global state. $A = \times_i A_i$ is the set of joint actions, where A_i is the set of actions for agent i. $P = S \times A \times S \rightarrow [0,1]$ is the state transition probability. $R = S \times A \rightarrow \mathbb{R}$ is the immediate reward. The complexity of Dec-MDP is nondeterministic exponential (NEXP) hard [1], hence learning is crucial.

2.4 The cooperative water monitoring scenario

We consider a water monitoring scenario, where several autonomous surface vessels are supervised by a human operator (see Fig. 1).

Each boat is capable of autonomous navigation and is equipped with an Android OS smartphone, custom electronics board, and sensor payload. The Android smartphone provides communication, through a wireless local area network, GPS, compass, and multi-core processor.

An Arduino Mega based electronics board receives commands from the Android phone over USB OTG and interfaces with the propulsion mechanism and sensor payload. The electronics board supports a wide variety of devices including acoustic doppler current profilers and sensors that measure electroconductivity, temperature, dissolved oxygen, and pH level. All sensor data is logged with time and location and streamed to a cloud based system that allows for real time visualization of the data.

The robot team is controlled from a nearby base station via wireless communitcation. The operator uses a plan monitoring tool based on colored petri nets [8]. Such plan monitoring framework allows to define high level team plans that specify sub-plans for each platform. A key feature of the framework is the ability to smoothly handle interrupt during plan execution, i.e., to change the sub-plans or part of the sub-plans that each platform is executing based on events that occur during the mission.

Such events include boats running out of battery power, possible loss of connection with the base station and traversing a dangerous area where GPS based navigation may fail. Each event may affect the normal behavior of the platforms and hinder their performance. Each event has a different probability of failure (see Table 1), where requests with the higher probability of failure are more crucial to receive the operator's attention. The relative values for the probability of failures associated with such events reflect knowledge gained from real deployments: it is extremely difficult for a boat that is running out of power to find an effective path that will ensure the platform to safely reach a recharge station. It is difficult to traverse a dangerous area using only autonomy, this of course is environment dependent but for standard environments such as lakes where boats are usually deployed this is a reasonable estimate; also consider that failures in this case does not refer to minor bumps into obstacles but situations that prevent the boat from moving autonomously (i.e., the boat being trapped by vegetation). Finally it is usually not an issue for a boat to operate without having a connection with the human operator but it could prove risky in some cases.

In this work we assume that the operator will always succeed to correctly handle an intervention request from the robotic platforms, however the operator can only process one request at a time, and this is why we use a queuing model to represent the interaction with the multi-robot system. Consequently, the time to service a specific request depends on how many requests are present in the queue and on the operator's speed (represented by the parameter μ).

3 Problem Formulation

Following previous works [5, 15], a central queue is provided to both the operator and the boats, where the oper-

Table 1 Different event types used in the experiments.

Event Type (E_j)	Prob. of Fail
Battery-Recharge (E_1)	0.9
Traversing-Dangerous-Area (E_2)	0.4
Losing-Connection (E_3)	0.2

ator can select one request at a time (i.e., in FIFO order) and assigns a specific sub-mission to resolve that request. A sub-mission is a plan specific recovery procedure, and this often requires a human interaction (i.e., the human directly selects which platforms should execute the interrupt sub-mission). We used three sub-missions, one for each class of requests, including: (i) sending a boat to the closest station to change/charge its battery. (ii) allowing/not-allowing a boat to go further (to the area that it might lose connection), and (iii) teleoperating a boat for traversing a specific area.

We assume that, whenever an event happens, the platform can detect the event. For example, a robot can perceive that its battery level is in a critical state, it must then decide whether to join the queue (i.e. sending the request and waiting for the operator) or balk (i.e., not sending the request) [2]. The consequences or costs of balking are problem specific. In our model, when a failure happens, the operator should spend more time to fix the problem, hence failure as a result of balking, increases the idle time of the system. Our goal is to minimize the idle time for the robot team, which is given by the time spent in the queue and the time required by the operator to address possible failures.

Our proposal is then to train the robots in a stationary environment (i.e., stationary distribution functions with fixed arrival rate and service time), so that the robots can learn appropriate balking policies. Then, by applying the learned policies in similar scenarios, they will be able to optimize the team objective. More specifically, we consider the following model in our domain: The state space $S = S_1 \times S_2 \times ... \times S_n$. n is the number of boats. The local state of each boat S_i is a tuple $\langle S_b, N_{tasks} \rangle$. N_{tasks} shows the number of remaining tasks of boat i. In this application domain, each task is a location that should be visited by a specific boat. S_b is the current internal state of boat i. More specifically $S_b \in \{E_j, \mathbf{W}aiting, \mathbf{F}ailed, \mathbf{A}utonomy\}$, where E_j refers to on of the request/event type in Table 1. For example, the state tuple of a boat when it has 3 tasks to finish and the event *Battery Recharge* occurs, would be $s = \langle E_1, 3 \rangle$. A_i is the set of actions for boat i where $A_i \in \{Join, Balk\}$. The reward function is designed to decrease the idle time (i.e. the time spent waiting for the operator).

[2] While this may be a significant challenge in some domains, this is not the focus of our work.

In general, there are two major approaches for learning in multi-robot scenarios [17]. The first approach is called team learning and uses a single learner to learn the behavior for the entire team. In contrast, the second approach uses multiple concurrent learners, usually one for each robot, where each learner tries to learn its behavior. Each of these methods has its own advantages and disadvantages which make it preferable in different domains [28, 17]. In particular, the major problems with team learning approach are the explosion of the state space (i.e., it keeps the states of the entire team), and the centralization of the learning approach that needs to access the states of all team members. Using the team learner in our application, the state space will be very large which decelerates the convergence to the optimal value. For example, for 5 boats with the above state representation, the state space will include more than one million states, hence requiring a prohibitively long time to estimate the optimal strategies for each state and action permutations. The main advantage of independent learners in our domain is that, this domain can be decomposed into subproblems (e.g. each boat holds its own state space) and each subproblem can be solved by one boat. In general, two main loosees arise in concurrent learning: credit assignment and non-stationary dynamics of the environment [17]. However, our application scenario has some special properties, that can be exploited to design a tractable model. First, the action selection at each step (i.e. when an event happens) only requires one agent to select either to join or balk. Hence, the reward can go directly to that agent. It is different from the situations, where all agents should decide at each step (i.e. joint actions), which results in the well-known credit assignment issue. However, when each boat considers only its local state without knowing the state of the queue, finding the optimal behavior for the team may become impossible, or the model may compute lower quality solutions. Therefore, we add the state of the queue to the local state of each boat, and then we use independent learners approach. To sum up, we consider three possible models:

Team Learner (TL): a team learner has access to the joint state of all robots which is $S = S_1 \times S_2 \times ... \times S_n$. When an event happens to a boat, the action $\langle Join, Balk \rangle$ for the corresponding boat will be selected and the state of the system will be updated. The update will only change the part of the state related to the corresponding boat. The Q-value of the team learner will be updated accordingly.

Independent Learners - Unobservable Queue (IL-U): an independent learner) is used for each boat. Each boat observes only its local state $S_i = \langle S_b, N_{tasks} \rangle$. In this model, each boat updates its local Q-values interacting to the system and receiving the reward.

Independent Learners - Observable Queue (IL-O): in this model, each boat in addition to observing its local state, has access to the size of the queue. The queue size shows the number of waiting boats inside the queue. The state representation of each boat in this model is: $S_i = \langle S_b, N_{tasks}, S_q \rangle$.

These models are different in their state representation, while the reward structure is the same for all of them:
(i) $R(S_t = S_i, A_t = Join) = R_S - (N_q \bar{\mu} + t_{serv})$.
(ii) $R(S_t = S_i, A_t = Balk) = R_F(\frac{\bar{\mu}}{\bar{\lambda}}) + N_q$; $if\, S_{t+1} = F$.
(iii) $R(S_t = S_i, A_t = Balk) = R_T$; $if\, S_{t+1} = A$.

The general rule for immediate reward in this model is the following: when a boat joins the queue it receives a positive reward R_S (this is because we assume that if the operator serves the request he/she will succeed) and a penalty that considers the size of the queue (N_q), the average service time ($\bar{\mu}$) and average time needed to resolve the request t_{serv}, this is to incentivize the minimization of the idle time (and specifically the waiting time in this case). When the boat balks, it may fail and in this case it receives a penalty R_F to represent the time that the operator will require to help the boat, this is multiplied by $\frac{\bar{\mu}}{\bar{\lambda}}$ to normalize for the speed of operator and event arrival rate ($\bar{\lambda}$). Finally, if the boat balks but it does not fail, a reward R_T will be assigned to the boat. The parameters R_S, R_F and R_T are domain specific parameters that must be tuned empirically, in our experiment we have $R_S = 1$, $R_F = -2$ and $R_T = 0.3$.

Notice that, the above function is expressed with respect to events that happen to specific agents (i.e., the agent joins or balks). Hence it may not be straightforward to see how such reward function can be used for the team learner approach that considers the joint state of all agents. The key point is that in our application scenario it is reasonable to assume that only one join or balk action will happen at any given time in the agent team (for more details see the discussion at the end of this section). Hence the reward structure can be directly used also for the team learner approach.

Finally, we use Q-Learning as the basis learning approach, while the same reward structure, same distribution functions for generating events and same distribution function for the service time are used for all models. We use Q-Learning because of its simplicity and good real time performance. Moreover, our goal is to propose the use of reinforcement learning in this novel context and not to provide a novel learning algorithm. In Q-Learning, each learner interacts with the environment (i.e. selects an action), receives the immediate reward and updates its state-action values (i.e. Q-values) as Eq.2:

$$Q_i(s_i,a_i) \leftarrow Q_i(s_i,a_i) + \alpha(r_i + \gamma \max_{a' \in A_i} Q_i(s',a') - Q_i(s_i,a_i)) \quad (2)$$

where r_i and s' are respectively the reward and the state observed by robot i after performing action a_i in state s_i; a' is the action in state s' that maximizes the future expected rewards; α is the learning rate and γ is the discount factor. Notice that, for Team Learner, there is only one Q-table (i.e., table of state-action values) to be updated, where, s_i refers to full state representation (i.e. state of all robots) and a_i is the action selected by learner for the specific robot with a request. This action will either add the event to the queue or not. As mentioned before, by exploiting specific feature of our domain, we can use an independent learner approach that is tractable and scalable.

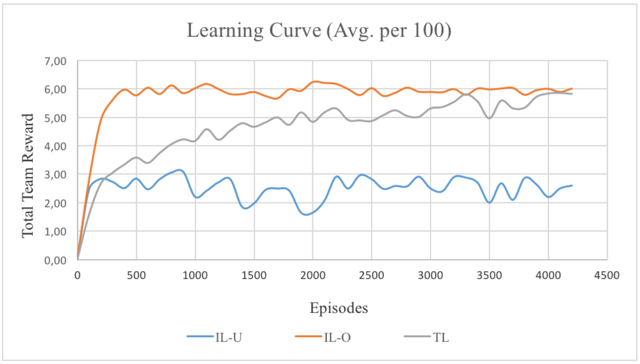


Fig. 2 Team accumulated reward in each episode of the learning phase (better viewed in color).

In the simulation, $\bar{\lambda}$ and $\bar{\mu}$ in the reward function are the same as λ and μ for generating the requests and service. However, for a field deployment, these two parameters should be estimated by considering the average number of events being generated during an interval as *lambda* (e.g. 20 events have been generated in 1 hour (or 60 minutes)), and the average time spent to fix each request as $\frac{1}{\mu}$ (e.g. 5 minutes to fix each request, hence 12 events can be fixed in 60 minutes).

The state of the queue, S_q, can be modified by robots' action (joining the queue) and the operator's action (leaving the queue). However, under the reasonable assumption that an arrival and a departure cannot happen exactly at the same time, only one entity can change the value of S_q at a time. Moreover, the possibility of having more than one event at the exact same time is very low. In particular, for our scenario, it is safe to assume that, the time to change the state of the queue, is much lower than the time for a new event arrival. Under this assumption, even if two events happen within a short time interval, the first one will affect the state of the queue before the second arrives, hence the other robots will base their decisions on the updated queue size.

4 Experimental Evaluation

4.1 Learning Phase

The learning phase of balking models starts by defining a list of locations (i.e., to be visited), and assigning those locations to boats. We consider 30 locations and 5 boats. Events, as in table 1, will be generated within an exponential distribution with parameter $\lambda = 0.25$. The operator's speed, for

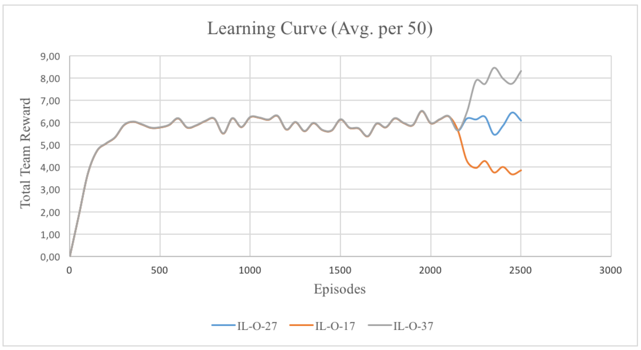


Fig. 3 Team accumulated reward in each episode of the learning phase (better viewed in color). From episode 2150, the service rate has been changed from 0.27 to 0.37 and 0.17.

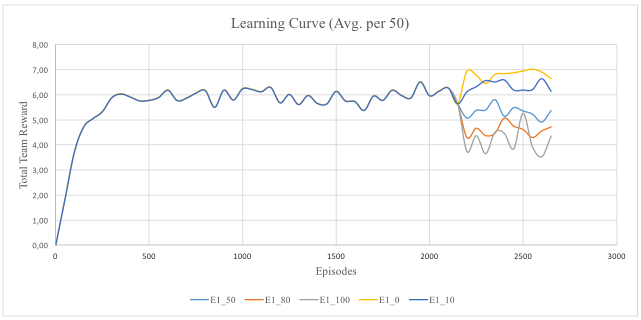


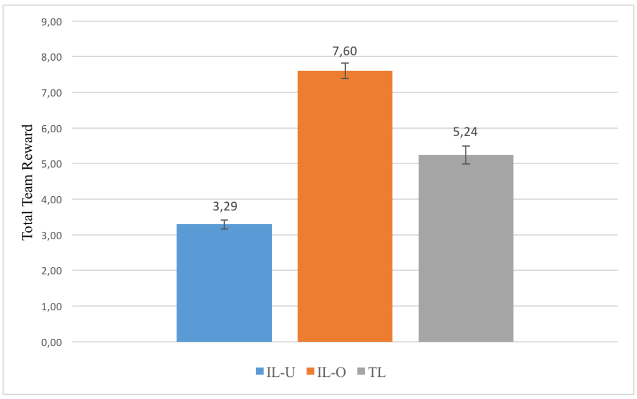
Fig. 4 Team accumulated reward in each episode of the learning phase (better viewed in color). After episode 2150, we vary the rate of each event type.

resolving a request is selected from an exponential distribution with parameter $\mu = 0.27$. An episode (i.e., a run of the algorithm beginning from a start state to a final state) ends after the system encounters 20 events. For action selection in our model, we use ε*_greedy* method with parameter $\varepsilon = 0.1$. Our algorithm uses the learning rate $\alpha = 0.1$ and discount factor $\gamma = 0.9$ throughout the experiments, which are tuned empirically. Each episode of the learning phase starts with all boats in their Autonomy state (i.e. they do not need the attention of the human operator), then with arrival rate λ an event may happen to one boat. We used values for λ and μ that define well the type of scenarios we are interested in, where boats can operate most of the time in autonomy, but frequently need user intervention.

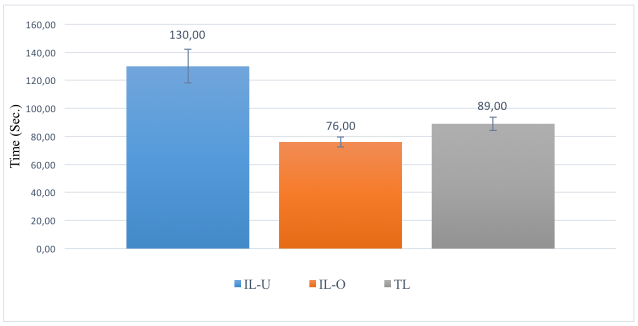
Fig. 2 shows the team rewards of each model, *TL*, *IL-U* and *IL-O*, during the learning phase. The oscillation in the reward is due to the fact that, the robots learn their policies by trying new potentially sub-optimal actions. The training time (the sum over 4000+ episodes) for each model was about 88-95 hours. As we expected, the convergence rate of *IL-O* is much faster than the *TL*, while they both reach a similar team reward. This is due to the larger state space of *TL* which needs more iterations to estimate the value for each state and action. Results also clearly show the importance of having access to the state of the queue to make better decisions. Since, the reward given to each action is related to the parameters λ and μ, we expect our policy to be dependent on these two parameters. Fig. 3 shows how *IL-O* adapts to changes in μ, where we increase and decrease its value by 40% during the learning phase. Possible variations of μ are important elements to consider because μ represents the speed of the operator and this is an important feature for the queuing policy. Since in general having a precise and accurate model of how fast the user can service different requests is not easy (particularly for robotics applications) we decided to use a standard model for service time (i.e., an exponential distribution with parameter μ) and show that our learning method can generalize to different operator speeds (i.e., different values of μ). In more detail, the graph in Fig. 3 shows that a sudden rise and drop of the team reward corresponds to changes in the value of μ, but then the system converges to a stationary state hence showing that the system is able to adapt the queuing policy to different operator's speeds. Fig. 4 shows another experiment, where we vary the ratio of event types during the learning phase. The events were generated with a uniform distribution up to episode 2150. After that, we vary the percentage of events from type 1(E_1), which has the higher probability of failure (see Table 1), while the rest of the events are uniformly distributed among E_2 and E_3. In more detail, we consider E_1-100, E_1-80, E_1-50, E_1-10 and E_1-0, where each shows the rate of E_1. Fig. 4 illustrates that, as there are more E_1, the system will gain less reward due to the increasing rate of failures. After several iterations, the learning curve becomes stationary.

4.2 Test Phase

After the learning phase, we run 30 simulation executing the policy learned previously. In this first experiment, we use the same values for λ and μ as used during the learning phase. Fig. 5(a) demonstrates the team reward for each learning models. A comparison on team reward between *IL-O* and *IL-U*, shows 56% gain for *IL-O*. Besides, a significant decrease (i.e. 40%) on average waiting time is shown in Fig. 5(b) when using *IL-O* rather than *IL-U*. One might expect the same reward value and idle time for *IL-O* and *TL*. However, the results on Fig. 5(a) and 5(b) show better performance values for *IL-O* than *TL*. Fig. 2 shows that towards the end of the learning phase *TL* reaches similar performance in terms of total team reward with respect to *IL-O*. However on the one hand *IL-O* always dominates and, more important, the reward value achieved by *TL* is much less stable with respect to the one achieved by *IL-O*. This results in the difference in performance observed in the test phase. The different behavior of the two methods is due to the fact that, the *IL-O* model keeps only the size of the queue or S_q (i.e. it does not consider which boats are waiting in the queue), while *TL* maintains the state of all boats which are



(a) Total team reward



(b) Total idle time

Fig. 5 (a) and (b) show the team performance (together with the standard error of the means) for three learning models

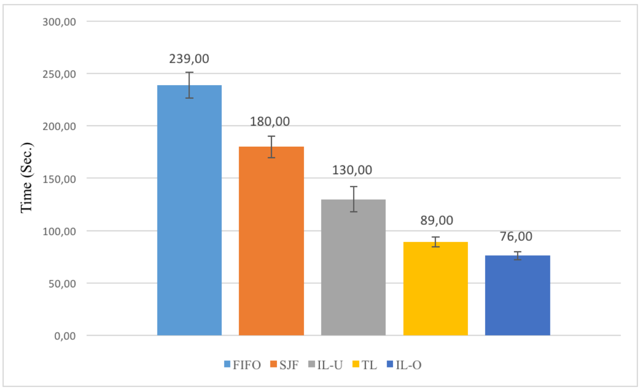


Fig. 6 Total idle time comparing balking models to non-balking models.

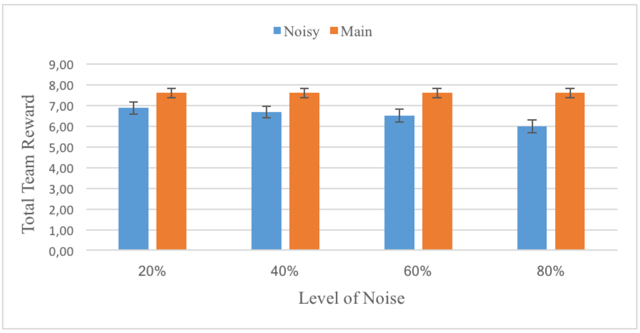


Fig. 7 Total team reward (together with the standard error of the means) for IL-O (main) with different levels of noise on λ and μ (noisy).

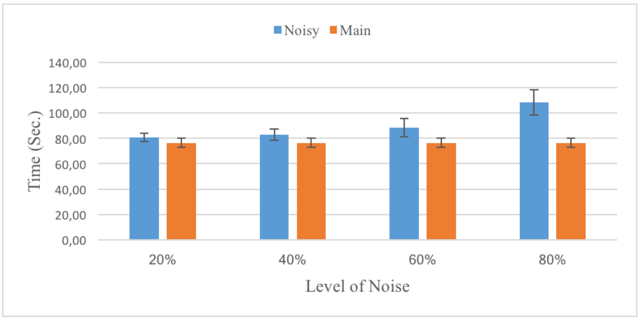


Fig. 8 Total idle time (together with the standard error of the means) for IL-O (main) with different levels of noise on λ and μ (noisy).

in their **W***aiting* state (i.e. $S_b = \mathbf{W}$). For example, whenever two boats waiting in the queue (assuming the other features, e.g., severity are the same), *IL-O* will map the state to $S_q = 2$, while *TL* will differentiate the states depending on which two boats are inside the queue. Since, the boats are homogeneous in our domain, *IL-O* results in better performance by abstracting away features that do not have a significant impact on the reward. This also makes *TL* to converge slower than *IL-O*, due to the larger state space of *TL* which needs more iterations to estimate the value for each state and action.

Next, we compare the behavior of queues with and without balking property (e.g., FIFO and SJF). For FIFO and SJF, we use the same event rate λ and service rate μ. In these two queuing models, boats always join the queue regardless of their request types and the queue size. Fig. 6 shows the team idle time for FIFO, SJF and three learning models. FIFO without balking, has the worst performance, since boats wait for the operator until he/she becomes available. In contrast, *IL-O* approach outperforms all other models. In more detail, it decreases the time up to 68% comparing to FIFO. In general, the results in Fig. 6 indicate that, using balking models significantly decreases the idle time of the team even though, some events may result in failures. This is acceptable in our domain, since the penalties for failures are not critical but only result in a finite increase of time.

To validate the noise sensitivity of our proposed model *IL-O*, we consider a set of experiments as follow. We consider adding the same level of noise, according to a uniform distribution, to both parameters λ and μ during the test phase. In more detail, the noise follows a uniform distributed with a range that is $X\%$ of the value of the parameter to which the noise is added. The value of X ranges in

$\{20, 40, 60, 80\}$. Fig. 7 shows the team reward and Fig. 8 shows the team idle time for different levels of noise. The results show that, the approach is able to cope with a significant amount of noise on both λ and μ.

As a final remark notice that both λ and μ can vary significantly in real deployments and these are key parameters for the cooperative queuing policy. However, a key outcome of our empirical results is that our learning approach can adapt to different values of μ (see Figure 3) and can cope with a significant amount of noise in the estimation of μ and λ (See Fig. 7 and 8). Hence, while clearly the absolute numbers for the performance indicators may vary, we believe that the key trends highlighted by our empirical evaluation (i.e., that the balking model performs better than standard queuing approaches and that our learning approach is effective) will remain valid for real deployments.

4.3 Discussion

The model for cooperative learning that we propose in this paper makes specific assumptions for what concerns the interactions of the system with the operator and the interactions among the robotic platforms. Such assumptions derive from the specific operation modalities of the cooperative water monitoring scenario described in Section 2.4. However, the idea of using a balking queue and the idea of learning the best cooperative balking policy using a model free approach can be applied to other domains, such as for example search and rescue applications, where robots should schedule operator intervention [5].

A first assumption relates to the interactions between the multi-robot system and the human operator. In particular, we do not take into account the possibility that the operator intervention might not be enough to correctly manage the unexpected event. Moreover, we consider a model for operator's skills and response time that does not depend on important factors related to the individual operator (e.g., her/his experience level) or to the specific situation (e.g., her/his stress level). These assumptions are not a major limiting factor for our water monitoring application, because the dynamics of the environment are usually rather slow allowing the operator to intervene successfully. Moreover, the events that we consider can be managed through rather simple interactions: the user receives an alarm through a software platform, and she/he will either change the current plan with another one or teleoperate the platform. However, for other application domains (and particularly for time-critical applications) such assumptions may not hold. A possible line of research to address this issue would be to have different models for different types of users so to consider their probability of failure and response time for different intervention requests. This would allow the robotic platforms to adapt their balking policy to different types of users hence improving the performance of the system.

A second assumption relates to the interactions among the team members. In particular, we assume that the state of the queue is always accessible by the robotic platforms. This assumption is reasonable in our application domain because we can rely on a robust communication infrastructure created by a dedicated device operating in an open area. Moreover, when robots stay within the range of communication allowed by the base station, they exchange mainly synchronization messages to allow for join plan execution, hence the bandwidth is not a limiting factor. Also, if one platform decides to go beyond the communication range it does not need to cooperate with its team mate until it comes back. However, in other domains, such as search and rescue, this assumption may not hold because the communication infrastructure is usually affected by the presence of debris or because interventions should be performed in indoor areas where communications can be blocked. In these cases, different learning methods should be employed. For example policy gradient methods may result in better performance than Q-Learning, because the learned policy can typically generalize to unobserved states [23]. Moreover, the policy gradient can learn stochastic policies that are more robust than deterministic ones, particularly for partially observable states.

5 Conclusions

In this paper, we propose the use of balking queue to model human-multi-robot interactions when the autonomy of robots allow them to decide whether to wait for the operator or not. We frame the problem as a Dec-MDP in which, each robot observes its local state and the state of the queue and cooperates with other agents to optimize the use of a shared queue. We apply independent Q-Learning to find these cooperative strategies in a water monitoring multi-robot simulation. We consider three different models (TL, IL-U, IL-O), and our results clearly show that an independent learner approach where the state of the queue is accessible to the platforms performs best. Furthermore, the empirical results related to the noisy estimation for λ and μ (Fig. 7 and 8), suggest that the approach is able to cope with a significant amount of noise.

Future directions in this area include modeling the situations where the boats can decide to leave the queue when the expected waiting time does not meet their requirements. This would require a change in the action space of each robot and a novel derivation of the reward function. Another interesting direction would be the use of different Multi-Robot Reinforcement Learning solutions such as policy gradient methods or in general function approximation methods to better capture the uncertainty in both learning and testing phases.

References

1. Bernstein, D.S., Givan, R., Immerman, N., Zilberstein, S.: The complexity of decentralized control of markov decision processes. Mathematics of operations research **27**(4), 819–840 (2002)
2. Bevacqua, G., Cacace, J., Finzi, A., Lippiello, V.: Mixed-initiative planning and execution for multiple drones in search and rescue missions. In: Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS'15, pp. 315–323. AAAI Press (2015)
3. Cacace, J., Caccavale, R., Finzi, A., Lippiello, V.: Attentional multimodal interface for multidrone search in the alps. In: 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 001178–001183 (2016). DOI 10.1109/SMC.2016.7844401
4. Chernova, S., Veloso, M.: Interactive policy learning through confidence-based autonomy. J. Artif. Int. Res. **34**(1), 1–25 (2009). URL `http://dl.acm.org/citation.cfm?id=1622716.1622717`
5. Chien, S.Y., Lewis, M., Mehrotra, S., Brooks, N., Sycara, K.P.: Scheduling operator attention for multi-robot control. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, Vilamoura, Algarve, Portugal, October 7-12, 2012, pp. 473–479 (2012)
6. Collins, J., Bilot, C., Gini, M., Mobasher, B.: Mixed-initiative decision support in agent-based automated contracting. In: Proceedings of the Fourth International Conference on Autonomous Agents, AGENTS '00, pp. 247–254. ACM, New York, NY, USA (2000)
7. Dorais, G.A., Bonasso, R.P., Kortenkamp, D., Pell, B., Schreckenghost, D.: Adjustable autonomy for human-centered autonomous systems. In: on Mars, in First International Conference of the Mars Society (1998)
8. Farinelli, A., Raeissi, M.M., Marchi, N., Brooks, N., Scerri, P.: Interacting with team oriented plans in multi-robot systems. Autonomous Agents and Multi-Agent Systems **31**(2), 332–361 (2017)
9. Goldman, C.V., Zilberstein, S.: Optimizing information exchange in cooperative multi-agent systems. In: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS '03, pp. 137–144. ACM, New York, NY, USA (2003)
10. Gromov, B., M. Gambardella, L., Di Caro, G.: Wearable multi-modal interface for human multi-robot interaction. In: in: IEEE International Symposium on Safety, Security, and Rescue Robotics, SSRR2016., pp. 240–245 (2016)
11. Gunderson, J.P., Martin, W.N.: Effects of uncertainty on variable autonomy in maintenance robots. In: IN WORKSHOP ON AUTONOMY CONTROL SOFTWARE, pp. 26–34 (1999)
12. Horvitz, E., Jacobs, A., Hovel, D.: Attention-sensitive alerting. In: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, UAI'99, pp. 305–313. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1999). URL `http://dl.acm.org/citation.cfm?id=2073796.2073831`
13. Hsieh, M.A., Cowley, A., Keller, J.F., Chaimowicz, L., Grocholsky, B., Kumar, V., Taylor, C.J., Endo, Y., Arkin, R.C., Jung, B., Wolf, D.F., Sukhatme, G.S., MacKenzie, D.C.: Adaptive teams of autonomous aerial and ground robots for situational awareness. Journal of Field Robotics **24**(11-12), 991–1014 (2007)
14. Kaminka, G.A., Frenkel, I.: Flexible teamwork in behavior-based robots. In: Proceedings of the 20th National Conference on Artificial Intelligence - Volume 1, AAAI'05, pp. 108–113. AAAI Press (2005)
15. Lewis, M., Chien, S.Y., Mehortra, S., Chakraborty, N., Sycara, K.: Task switching and single vs. multiple alarms for supervisory control of multiple robots. In: International Conference on Engineering Psychology and Cognitive Ergonomics, pp. 499–510. Springer (2014)
16. Naor, P.: The regulation of queue size by levying tolls. Econometrica **37**(1), 15–24 (1969). URL `http://www.jstor.org/stable/1909200`
17. Panait, L., Luke, S.: Cooperative multi-agent learning: The state of the art. Autonomous agents and multi-agent systems **11**(3), 387–434 (2005)
18. Raeissi, M.M., Farinelli, A.: Learning queuing strategies in human-multi-robot interaction. In: Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '18, pp. 2207–2209. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2018)
19. Rosenfeld, A., Agmon, N., Maksimov, O., Azaria, A., Kraus, S.: Intelligent agent supporting human-multi-robot team collaboration. In: Proceedings of the 24th International Conference on Artificial Intelligence, IJCAI'15, pp. 1902–1908 (2015)
20. Rosenthal, S., Veloso, M.: Using symbiotic relationships with humans to help robots overcome limitations. In: Workshop for Collaborative Human/AI Control for Interactive Experiences (2010)
21. Scheutz, M., Kramer, J.: Reflection and reasoning mechanisms for failure detection and recovery in a dis-

tributed robotic architecture for complex robots. In: Robotics and Automation, 2007 IEEE International Conference on, pp. 3699–3704. IEEE (2007)
22. Stoica, A., Theodoridis, T., Hu, H., McDonald-Maier, K., Barrero, D.: Towards human-friendly efficient control of multi-robot teams. In: Proceedings of the 2013 International Conference on Collaboration Technologies and Systems, CTS 2013, pp. 226–231 (2013)
23. Sutton, R.S., McAllester, D.A., Singh, S.P., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: Advances in neural information processing systems, pp. 1057–1063 (2000)
24. Suzanne Barber, K., Goel, A., Martin, C.E.: Dynamic adaptive autonomy in multi-agent systems. Journal of Experimental & Theoretical Artificial Intelligence **12**(2), 129–147 (2000)
25. Tambe, M.: Towards flexible teamwork. J. Artif. Int. Res. **7**(1), 83–124 (1997)
26. Valada, A., Velagapudi, P., Kannan, B., Tomaszewski, C., Kantor, G., Scerri, P.: Development of a Low Cost Multi-Robot Autonomous Marine Surface Platform. Springer Berlin Heidelberg (2014)
27. Wang, J., Lewis, M.: Human control for cooperating robot teams. In: Proceedings of the ACM/IEEE International Conference on Human-robot Interaction, HRI '07, pp. 9–16. ACM (2007)
28. Xuan, P., Lesser, V.: Multi-agent policies: From centralized ones to decentralized ones. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 3, AAMAS '02, pp. 1098–1105. ACM (2002)
29. Zhang, Y., Narayanan, V., Chakraborti, T., Kambhampati, S.: A human factors analysis of proactive support in human-robot teaming. In: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 3586–3593 (2015). DOI 10.1109/IROS.2015.7353878