OMCoRP: An Online Mechanism for Competitive Robot Prioritization

Sankar Narayan Das[1*], **Swaprava Nath**[2], **Indranil Saha**[2]
[1] Accenture Labs, India
[2] Indian Institute of Technology Kanpur, India
sankar.c64@gmail.com, {swaprava, isaha}@cse.iitk.ac.in

Abstract

We propose a collision-avoiding mechanism for a group of robots moving on a shared workspace. Existing algorithms solve this problem either (a) in an *offline* manner using the source-destination information of all the robots or (b) in an online manner with *cooperative* robots. We take a paradigm shift to the setting with *competitive* robots, that may strategically reveal their urgency of reaching the destinations and design an *online* mechanism that takes decisions on-the-fly, reducing the overhead of an offline planning. We propose a mechanism, named **OMCoRP**, in this setting that ensures truthful revelation of the robots' priorities using principles of economic theory and provides locally efficient movement of the robots. It is free from collisions and deadlocks, and handles dynamic arrival of robots. In practice, this mechanism gives a smaller delay for robots of higher priority and scales well for a large number of robots without compromising on the path optimality too much.

1 Introduction

Collision avoidance is a central problem in various multi-agent path planning applications. The problem has been provided different solutions in different paradigms (Snape et al. 2010; Hennes et al. 2012; Chen et al. 2017; Desai et al. 2017; Gavran, Majumdar, and Saha 2017, e.g.). In this paper, we focus on multiple *dynamically arriving* and *independently controlled* robots that have different *priority requirements* for moving from their sources to destinations using a track network. The problem is motivated by many commercial settings where the robots are controlled by independent operators, e.g., in autonomous vehicle movements or shared warehouses for goods dispatch, and have strict deadlines of delivery leading to different levels of urgency. In such settings, robots can potentially *manipulate* for a prioritized scheduling. In this paper, we leverage the potential of online mechanism design to ensure truthful, class-prioritized dispatch of robots along with other desirable properties like collision and deadlock avoidance.

[*]The author acknowledges the support of a postdoctoral research grant from IIT Kanpur.

1.1 Our Approach and Results

We propose an online mechanism that decides the movements when multiple robots come to a near-collision situation, i.e., the robots try to simultaneously access less amount of space. The competitive robots in our setting engage in a *game*[1] and bid the amount they are willing to pay to get access to those spaces. In our protocol, the robots that 'win' the game get access to move, but for a payment. Since the true urgency of the robots are their private information, the challenge in designing a protocol that picks the 'truly' deserving robots as the winners of this game requires *truthful mechanism design* approach.

To this end, we consider a quasi-linear payoff model (Shoham and Leyton-Brown 2008, Chap 10) for the robots and propose **O**nline **M**echanism for **Co**mpetitive **R**obot **P**rioritization (**OMCoRP**), which decides the priority plan online during a collision scenario. We show that **OMCoRP** is *collision-free, locally deadlock-free*, and *robust against entry-exit* (Claim 1). Every competitive robot reveals its private information truthfully under **OMCoRP** (Theorem 1), does not have a positive surplus of money, and is *locally efficient* (Theorem 2).

Detailed experiments (§6) show that **OMCoRP** (1) scales well with large number of robots, (2) takes much less running times compared to the state-of-the-art offline motion planning algorithms like M* (Wagner and Choset 2011) and prioritized planning (van den Berg and Overmars 2005), (3) does differentiated prioritization of different classes of robots, and (4) handles dynamic arrivals smoothly. The experimental results show that even if **OMCoRP** were to be implemented in a *cooperative* environment (as opposed to the *competitive* environment, for which it is designed), it would still perform reasonably well as an online collision-avoiding algorithm.

We have simulated our mechanism for up to 500 robots in Python experimental setup and up to 10 TurtleBots[2] using ROS (Quigley et al. 2009). Successful simulation in ROS promises that the proposed mechanism can be implemented in a real multi-robot system.

[1]A *game* is a strategic interaction between multiple self-interested agents.

[2]http://turtlebot.com/

1.2 Related Work

Robots sharing the same track network (Guizzo 2008; Bogue 2016; Wulfraat 2016) are prone to colliding with each other. To avoid the collision, primarily *three* different approaches are employed in the literature.

In the first approach, static multi-robot planning algorithms are employed to generate the collision-free paths for all the robots offline (Erdmann and Lozano-Perez 1986; van den Berg and Overmars 2005; Yu and LaValle 2013; Turpin, Michael, and Kumar 2014; Wagner and Choset 2011; Saha et al. 2014, 2016, e.g.). However, (a) the *computation time* for generating path plans for a large number of robots may be prohibitively high, and (b) they cannot deal with the *dynamic arrival* of new robots to the system without recomputing the whole plan.

In the second approach, the robots independently generate their trajectories offline without the knowledge about the trajectories of the other robots (Azarm and Schmidt 1997; Chun, Zheng, and Chang 1999; Jager and Nebel 2001; Pallottino, Scordio, and Bicchi 2004; Olfati-Saber, Fax, and Murray 2007; Hoffmann and Tomlin 2008; Purwin, D'Andrea, and Lee 2008; Velagapudi, Sycara, and Scerri 2010; Snape et al. 2010; Desaraju and How 2012; Chen et al. 2017, e.g.). Hence, the trajectories of the robots are not collision-free, but are resolved online in a decentralized manner through information exchange among the potentially colliding robots, assuming that the robots will *cooperate* with their movements. If the simultaneous movements of the robots are not possible, the robots run a distributed consensus algorithm to find a collision-free plan.

In the third approach, robots bid for their makespan or demand for resource in an auction-like setting (Lagoudakis et al. 2005; Bererton, Gordon, and Thrun 2004; Nunes and Gini 2015; Calliess, Lyons, and Hanebeck 2011). Algorithms designed for such setups do not allow robots to be owned and controlled by independent agents, and therefore do not ensure that these agents *bid* their privately observed information (makespan or demand of resource) *truthfully*. However, the combinatorial auction reduction of the MAPF problem (Amir, Sharon, and Stern 2015) does satisfy truthtelling, but it is centralized and therefore has the same limitations of time complexity and dynamic arrival. The other strand of literature (Takei et al. 2012; Dhinakaran et al. 2017) on non-cooperative robots consider other robots as dynamic obstacles and solve computationally hard mixed integer programs in a centralized manner.

2 Problem Setup

Define $N := \{1, \ldots, n\}$ to be the set of robots that are trying to travel from their sources to destinations over a directed graph $G = (V, E)$, where V is the vertex set and E is the edge set. We assume that each vertex has at least one outgoing edge. Time is discrete and is denoted by the variable t. Every edge in the graph is partitioned into cells where a robot can stand at any given time step. Every cell is uniquely numbered with S being the total number of cells. We represent the k-th cell by x_k, $k \in \{1, \ldots, S\}$. We call every vertex having an in-degree of two or more an *intersection* point, since robots moving into such vertices cannot guarantee to avoid a collision without coordinating with each other. An intersection vertex is a roundabout having a fixed number of cells, which we call the *capacity* of the intersection. Each cell has a unique identity and the list of all cells, with the intersection nodes specially marked, are shared beforehand to all the robots. Therefore, every robot knows the set of intersection nodes a priori. Every roundabout can hold robots equal to the number of cells at any given time and the movement of all the robots is unidirectional (e.g., counter-clockwise). An example of an intersection vertex with four incoming and four outgoing edges is illustrated in Fig. 1. Note that *any* directed graph with slotted directed edges can be equivalently represented with a graph having directed edges and vertices with roundabout structure.

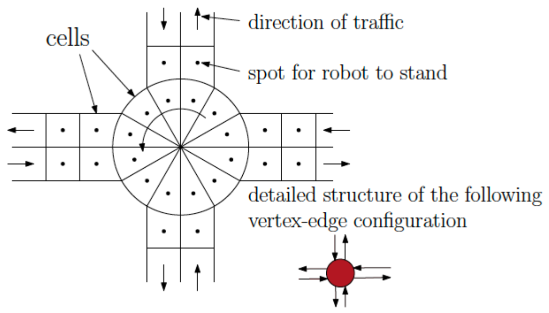

cells
direction of traffic
spot for robot to stand
detailed structure of the following vertex-edge configuration

Figure 1: Illustration of a vertex with four incoming and four outgoing edges.

In an online traversal plan of the robots, every robot decides its path offline based on its private information and through communication with a *local intersection manager* (LIM), i.e., a trusted automated intermediary at every intersection. The role of an LIM is to collect the robots' (near that intersection) urgency information and coordinate a local movement plan when the robots reach near the intersection. Let the path of robot $i \in N$ be denoted by $P_i := (x_{k_1}, x_{k_2}, \ldots, x_{k_{l(i)}})$, where x_{k_j}'s are the cells of the graph leading from the source to the destination of i and $l(i)$ is the length of the path. A *collision* occurs if more than one robot simultaneously move into a cell. We assume that a collision is infinitely costly that each robot wants to avoid. Collision can occur (a) at an intersection where two robots intend to move to the same empty cell, or (b) on a cell on an edge where a robot is stationary and another robot moves into the same cell. The current location of robot i at t be denoted by $\ell_i(t)$, and the next intended location by $n_i(t)$.

We assume that these robots are independently controlled (or owned) by agents who have no information about other robots' locations, sources, destinations, and priorities. Each robot has a deadline to reach its destination, which is encoded into its *value* at each time step, i.e., a monetary gain when the robot is allowed to move in a time step. The priority of a robot is determined by this number since a robot with a high value indicates that it has a high urgency to reach its destination, e.g., robots carrying priority shipping items.

We assume that a robot's value in each time step is calculated internally by the entity that controls it by considering several factors, e.g., its proximity to the deadline, the type of objects it is carrying (high, if it is carrying a priority shipping item), and all such factors are consolidated into a real number $v_i(t)$ for robot i if it can move to its next location $n_i(t)$ at time $t+1$. If the robot does not move at t, the value $v_i(t)$ is zero[3]. The value $v_i(t)$ of robot i is measured in the unit of money that it is willing to pay for its movement at $t+1$.

We emphasize that in this paper, we have used the intersection of *four* incoming and outgoing edges (Fig. 1) as a running example for simplicity and also because this structure is widely used in real applications, e.g., in warehouses of Amazon[4] or Alibaba[5]. However, our results are perfectly general for any directed graph with vertices having a traffic roundabout and a pre-defined rule of movement (e.g., counterclockwise in the example of Fig. 1).

In this paper, our objective is to design an *online mechanism* that is formally discussed in the following section.

3 Online Mechanisms

In an online mechanism, every robot independently finds the shortest path from its source to destination and follows it. These paths may not be collision-free. Therefore, an online mechanism needs to resolve the two aforementioned possible types of collision when the robots are mobile. The collision between a stationary robot with another that moves in the same cell on an edge can be easily resolved by equipping every robot with a sensor that can sense the movement of the other robot(s) in front of it. High accuracy range-finding sensors like LiDAR[6] provide autonomous robots with such capability. The main task of collision avoidance and prioritization therefore happens at the intersections.

We consider online mechanisms in this setting that aim to achieve an *efficient*[7] (Shoham and Leyton-Brown 2008, Chap 10) plan of movement. When robots compete for prioritized movements, a desirable mechanism prioritizes the robots with higher values over that with lower values – breaking ties arbitrarily. We provide a formal definition of 'efficiency' in Section 4.

Since the robots are independently controlled, the value information, $v_i(t), i \in N$, are private to the agents. Collision mitigation in such scenarios needs a *mechanism* which asks each robot to report their values at every collision scenario and plans the movement. Competitive robots can *overbid* its value to ensure that it is prioritized in every collision scenario.

This is precisely where our approach using the ideas of mechanism design (Börgers 2015) is useful. In mechanism design theory, it is known that in a private value setup, if the mechanism has no additional tool to penalize overbidding, only degenerate mechanisms, e.g., *dictatorship* (where a pre-selected agent's favorite outcome is selected always), are truthful (Gibbard 1973; Satterthwaite 1975). This negative result holds irrespective of whether agents' preferences are *ordinal* (representable as an order relation over the outcomes) or *cardinal* (agents have a real number to represent the intensity of the preference). Note that in our setup, the robot's preferences are cardinal. A complementary analysis by Roberts (1979, Thm 7.2) shows that a dictatorship result reappears under certain mild conditions in a *quasi-linear* setting (which is our current setting and is formally defined later), unless monetary transfers are allowed. Money, in these settings, is used as a means of transferring value from an agent to another, and it is shown to help by unraveling the true values of the agents. In this paper, we, therefore, use monetary transfers among the agents to ensure truthful value revelation at every round.

Robot payoff model: At every time step t, given the current position of the robots, denote the set of feasible next-time step configurations by $A(t+1)$. Hence, for every feasible configuration $a \in A(t+1)$, robot i's valuation is given by $\mathbf{val}_i(a, v_i(t)) = v_i(t)$ if robot i is allowed to move to $n_i(t)$ under a, and zero otherwise. The mechanism also recommends robot i to pay $p_i(t)$ amount of money. The net payoff of robot i is given by the widely used *quasi-linear* formula (Shoham and Leyton-Brown 2008, Chap 10)

$$\mathbf{val}_i(a, v_i(t)) - p_i(t). \tag{1}$$

A strategic robot reports its private information $v_i(t)$ to maximize its net payoff. We assume that the robots are *rational*, i.e., choose their actions to maximize their payoff. A mechanism in such a setting is defined as follows.

Definition 1 (Mechanism) *A mechanism* $(\mathbf{f} := [f_t]_{t=1,2,\ldots}, \mathbf{p} := [p_{i,t}]_{i\in N, t=1,2,\ldots})$ *is a tuple which decides the* allocation *and* payment *for every robot at every time step t for the reported values of the players denoted by* $\hat{v}_1(t), \ldots, \hat{v}_n(t)$. *Here the allocation function at time t is given by* $f_t : \mathbb{R}^n \mapsto A(t+1)$ *that decides the next location of every robot* $i \in N$, *and* $p_{i,t} : \mathbb{R}^n \mapsto \mathbb{R}$ *denotes the payment made by the robot. Hence,* $f_t(\hat{v}_1(t), \ldots, \hat{v}_n(t))$ *and* $p_{i,t}(\hat{v}_1(t), \ldots, \hat{v}_n(t))$ *denote the allocation and payment of robot i respectively at time step t.*

Local intersection management: We assume that a mechanism $(\mathbf{f}, \mathbf{p})$ is implemented by the LIMs at every intersection. The robots that are either inside or near an intersection participate in the mechanism and bid their values to the LIM. At the beginning of each time step t, an LIM collects the bids of the local robots that sent their bids until t, decides the allocation f_t and collects the payments according to $\mathbf{p}_t$. Any bid that comes after that is considered at step $t+1$, and the LIM provides an option to that robot to update its bid before $t+1$ [8]. That way the LIM synchronizes the robots which are not carrying a synchronized clock. In this paper, we con-

[3]We assume this for simplicity and our results hold even when the value is non-zero and has the same time complexity.

[4]Video: https://youtu.be/Ox05Bks2Q3s?t=20

[5]Video: https://youtu.be/jwu9SX3YPSk?t=37

[6]https://www.geospatialworld.net/blogs/why-lidar-is-important-for-autonomous-vehicle/

[7]In microeconomic theory, the alternative that maximizes the *sum* of the values of all the agents is called 'efficient' (see (Börgers 2015)).

[8]This is because if a robot submits its bid at $t+\epsilon$, with $\epsilon > 0$ small, it may have a different valuation at $t+1$.

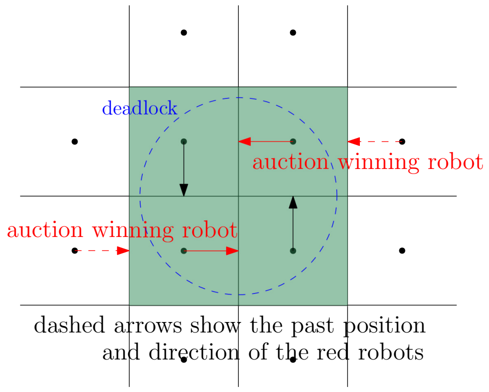

deadlock
auction winning robot
auction winning robot
dashed arrows show the past position
and direction of the red robots

Figure 2: Example of a deadlock scenario. Arrows denote their intended direction of move.

sider allocation and payment functions to be stationary, i.e., independent of t.

A naïve online mechanism: Consider a mechanism which allows the highest bidder to move whenever two or more robots intend to enter a common cell. This allocation rule can be made truthful by adding the following *payment rule*. The robot that is prioritized pays the amount equal to the bid of the second highest bidding robot that stops at that time. This is the well-known *second price auction* (Vickrey 1961), which is known to be truthful. Clearly, the robots under this mechanism successfully avoid collision in an online manner. But, it can lead to the following *deadlock* scenario. Suppose the intersection is of four full-duplex paths with four cells at the intersection (Fig. 2). If there are two robots at the intersection and two more are attempting to enter, and both the entering robots win the auction for those cells, it will lead to the four cells at the intersection occupied with four robots. If these robots' next locations are the current locations of the robots already in the intersection, none of them can move under this mechanism (see Fig. 2), leading to a deadlock.

To avoid such a deadlock, a mechanism cannot allow more than $(m-1)$ robots to enter an intersection of capacity m. We propose the online mechanism **OMCoRP** (**O**nline **M**echanism for **Co**mpetitive **R**obot **P**rioritization) to resolve the issue of deadlock, and also ensure prioritized treatment of truly deserving robots. It considers the quasi-linear payoff model and designs the allocation of cells and payments to obtain several desirable properties.

3.1 Description of OMCoRP

For a given graph G, let the set of indices of the intersection vertices be represented by K. Denote the cells of intersection vertex $k \in K$ by $I_k(V)$. **OMCoRP** has two components: (a) the primary component runs at the LIMs at every intersection; it listens to the messages sent by the robots either inside or attempting to enter the intersection and decides the allocation and payment for each of them at every time step, and (b) a recommendation for a collision-free travel on an edge. Part (b) is rather straightforward, where each robot is asked to STOP and remain stationary in its current cell, or GO to its intended cell. We assume that in addition to taking actions STOP and GO, the robots are also capable of taking an action FOLLOW. In this action, a robot follows the robot that is currently obstructing its movement (call it the *leader* robot), and if the leader moves then the follower also moves in the same time step. Hence, in the remaining part of this section, we will discuss only part (a) of **OMCoRP**.

OMCoRP decides when the robots enter an intersection. Consider intersection k. At every time step t, the LIM at k finds the set of robots $N_k(t)$ that are either inside the intersection or are attempting to enter the intersection. From the information shared by all the robots $(\ell_i(t), n_i(t), v_i(t)), i \in N_k(t)$, the mechanism allows the LIM to find the feasible next-step configurations $A_k(t+1)$ at that intersection. The feasible configurations ensure that there are no more than $(m-1)$ robots in the intersection of capacity m at any time step t. The proposed mechanism picks the configuration

$$a_k \in \operatorname*{argmax}_{a \in A_k(t+1)} \sum_{i \in N_k(t)} \mathbf{val}_i(a, v_i(t)). \tag{2}$$

The configuration maximizes the sum of the values of all the robots that are either inside or are entering the intersection. Denote this maximizer by $a_k^*(t+1)$. Similarly, we can define a *sum-value maximizing* configuration excluding robot i as follows.

$$a_k^{N_k(t)\setminus\{i\}} \in \operatorname*{argmax}_{b \in A_k^{N_k(t)\setminus\{i\}}(t+1)} \sum_{j \in N_k(t)\setminus\{i\}} \mathbf{val}_j(b, v_j(t)). \tag{3}$$

Denote this sum-value maximizing configuration except i by $a_k^{*,N_k(t)\setminus\{i\}}(t+1)$. Define the payment as the difference

$$p_i(t) := \sum_{j \in N_k(t)\setminus\{i\}} \mathbf{val}_j(a_k^{*,N_k(t)\setminus\{i\}}(t+1), v_j(t)) - \sum_{j \in N_k(t)\setminus\{i\}} \mathbf{val}_j(a_k^*(t+1), v_j(t)). \tag{4}$$

We are now ready to present our proposed mechanism **OMCoRP**. For the LIM of intersection k at time t, it is described in Algorithm 1. The mechanism is repeated at every t at every intersection $k \in K$ until each robot reaches its destination.

OMCoRP is the standard Vickery-Clarke-Groves (VCG) mechanism (Vickrey 1961; Clarke 1971; Groves 1973) run among the robots in $N_k(t)$ at every time step t. It is known that repeated version of the VCG mechanism may not satisfy all the properties of the static version. However, we discuss later in this section why **OMCoRP** induces a fresh instance of the VCG mechanism in every time step in our setup and therefore differs from the repeated VCG.

Note that in the algorithm, we have assumed that the current and next locations $\ell_i(t), n_i(t)$ of robot $i \in N_k(t)$ are known to the LIM. This can either be measured or asked from the robot i itself. The robots will be reporting it truthfully since if this is misreported, the LIM will plan the movements according to the misreported locations. This may lead to a plan where some other robot is suggested to move into a location which is the current location of the misreporting robot. This will lead to a collision. Since the robots cannot

Algorithm 1 **OMCoRP** run by the LIM at $I_k(V)$ at time t

```
1: Input: For every robot $i \in N_k(t)$, the current and future cells $\ell_i(t), n_i(t)$, and reported value $\hat{v}_i(t)$
2: Output: Decisions for every robot $i \in N_k(t)$ to STOP/GO/FOLLOW
3: Compute $a_k^*(t+1)$ (Equation (2)) and suggest the robots $i \in N_k(t)$ to STOP/GO/FOLLOW according to that recommendation in time $t+1$
4: Charge $p_i(t)$ (Equation (4)) to every $i \in N_k(t)$
5: Collect this money and distribute equally to all $j \in N \setminus N_k(t)$
```

communicate with each other in our model and that the collisions are infinitely costly, the best response for every robot is to report the locations truthfully. Hence in the rest of the paper, we will consider that the robots can misreport only in their valuations.

Also in Step 5 of Algorithm 1, the consolidated payment collected by the LIM at intersection k at t from all the robots at that intersection is distributed equally to the robots that were not part of intersection k at t. Therefore, **OMCoRP** does not accumulate money from the agents.

3.2 Few Observations on the Mechanism

In this section, we make a few observations on the purpose of the LIMs and the difference of **OMCoRP** with a repeated VCG mechanism.

Efficacy of the LIMs: The whole computational burden of finding a collision-free path in **OMCoRP** is divided between the LIMs and the robots unlike that of a central planner in centralized motion planning algorithms. However, the local computations at the LIMs and the robots are minimal and they satisfy certain important properties as we will see in the following sections.

OMCoRP does not require the robots to have a synchronized clock. Each robot either FOLLOWs the robot in front of it (if present) or continues its movement according to the shortest path plan when on the edges of the graph. When robot i reaches an intersection, it stops and transmits the requisite bid to the LIM and waits for the LIM's signal before moving. This makes **OMCoRP** a locally synchronized online mechanism.

It is worth discussing the need of the LIMs, since the local decisions of movement at the intersections could have been coordinated by the robots themselves in a decentralized manner. They could exchange messages of their current, intended future locations, and valuations, and compute the local plan of movement at every robot's end. There are two disadvantages of this approach. In this mode, each robot repeats the computation held at the LIM. But most importantly, the VCG mechanism employed here is designed for simultaneous actions of the players, which keeps the communication among the players and computation by the LIM simpler. It does not work if the robots can take sequential actions (picks action after observing the action of others). In practice, for the decentralized approach (a) perfect synchronization among the messages of the robots is needed which is difficult to achieve in practice, and (b) the competitive robots may not synchronize deliberately since the last mover has advantages in sequential actions. Hence, we need a trusted arbitrator like the LIM that receives the information of the robots and communicates the movement plan, but does not reveal the reported information to the other robots.

Difference with the repeated VCG: The LIMs do not share the reported information of the robots to one another, hence a robot can only learn the movement of the other robots and infer their relative valuations across the rounds. E.g., if robot A moves in a round where robot B is kept stationary, then B learns that A had a higher valuation. However, if both A and B move or remains stationary in a round, they would not learn their relative valuations as they only communicate with the LIM. If we focus on a pair of robots taking part in **OMCoRP** at an intersection at t, they may or may not have competed for a cell in the intersection ('compete' means that in a feasible allocation both of them cannot be allowed to move). If they competed, only then they could learn about the relative valuation of the other robot. But it is easy to see that under **OMCoRP** if the pair of robots compete in a time step t, they do not again compete in any future time step $t' > t$. This is because if two robots cannot simultaneously be a part of a feasible allocation in any time step, one of them will move under this mechanism, and in future time steps, the same pair of robots will not compete again for a cell in the intersection. This is how **OMCoRP** avoids the repeated interaction of the robots.

Before advancing to the desirable properties, here is the summary of the features and assumptions of **OMCoRP**. (1) All robots are locally synchronized in time via an LIM, which receives the messages from the local robots and creates the plan of movement online. The robots do not need to carry a synchronized clock for **OMCoRP**. (2) The model and results are for any arbitrary directed (direction showing the flow of traffic) graph. The examples and experiments are on specific graphs that represent real-world configurations. (3) Each robot has a geometry that fits well within a cell and does not collide with a robot in a different cell. (4) Robots are equipped with sensors that allow them to execute the FOLLOW action (see discussion in the beginning of Section 3).

4 Design Desiderata

In an online robot path planning algorithm, the agents choose their own route from the source to the destination. The collision avoidance mechanism ensures a protocol that is applied locally at a potential colliding scenario. The property desirable in such a setting is a *locally efficient* prioritization.

Definition 2 (Local Efficiency) *A robotic collision avoidance mechanism* $(\mathbf{f}, \mathbf{p})$ *is* locally efficient *if for every time t, every intersection* $k \in K$*, it chooses an allocation that maximizes the sum value of all the robots in* $N_k(t)$*. Formally, it picks* $\forall t, \forall k \in K$

$$f(v_i(t), i \in N_k(t)) \in \operatorname*{argmax}_{a \in A_k(t+1)} \sum_{i \in N_k(t)} \mathbf{val}_i(a, v_i(t)).$$

However, in a multi-agent setting, $v_i(t)$'s of the robots are unknown to the mechanism, which can only access the reported values $\hat{v}_i(t)$'s. Therefore, the following property ensures that the robots are incentivized to 'truthfully' reveal these information.

Definition 3 (Dominant Strategy Truthfulness) *A mechanism* $(\mathbf{f}, \mathbf{p})$ *is* truthful in dominant strategies *if for every* t, $v_i(t), \hat{v}_i(t), \hat{v}_{-i}(t)$[9]*, and* $i \in N$

$$\begin{aligned}&\mathbf{val}_i(f(v_i(t), \hat{v}_{-i}(t)), v_i(t)) - p_i(v_i(t), \hat{v}_{-i}(t)) \\ &\geqslant \mathbf{val}_i(f(\hat{v}_i(t), \hat{v}_{-i}(t)), v_i(t)) - p_i(\hat{v}_i(t), \hat{v}_{-i}(t)).\end{aligned}$$

The inequality above shows that if the true value of robot i is $v_i(t)$, the allocation and payment resulting from reporting it 'truthfully' maximizes its payoff *irrespective of the reports of the other robots* (hence the name dominant strategy).

Since we consider mechanisms with monetary transfer, an important question is whether it generates a surplus amount of money. The following property ensures that there is neither surplus nor deficit.

Definition 4 (Budget Balance) *A mechanism* $(\mathbf{f}, \mathbf{p})$ *is* budget balanced *if for every* t *and* $(v_i(t), v_{-i}(t))$, $\sum_{i \in N} p_i(v_i(t), v_{-i}(t)) = 0$.

In the context of online robot path planning, a mechanism that is robust against robot failures is highly desirable. Also, it is desirable if a robot can start its journey when other robots are already in motion, and the mechanism does not need other robots to re-compute their path plan.

Definition 5 (Entry-Exit Robustness) *A mechanism* $(\mathbf{f}, \mathbf{p})$ *is* robust against entry or exit *of the robots if (a) it allows newly arrived robots to join immediately, and (b) the properties satisfied by the mechanism at a given time step with the existing robots are unaffected by an addition or deletion of a robot.*

Offline, centralized collision avoidance mechanisms that compute the paths for all robots and recommend those paths to the robots are not robust against entry or exit. With every addition or deletion of a robot, the plan has to be recomputed. On the other hand, entry-exit robustness is not a consequence of a mechanism being online. Online-ness only implies that the decisions are taken on on-the-spot by either the robots or the LIMs. It does not restrict the way in which they interact with each other. Based on the interaction, the plan may not be robust against entry-exit, as the following example shows.

Example 1 (Online but not Entry-Exit Robust)
Consider the following online version of the prioritized planning (PP) algorithm (van den Berg and Overmars 2005). Before beginning to move, the robots send their identities on a common channel (assume that there is a wired broadcast channel connecting all the starting positions) and determine their relative priorities, but this common channel is not available when they are on the move. The robot with the highest priority computes its path and broadcasts it (the priority order is fixed beforehand and is a common knowledge of all the robots). After listening to that plan, the second highest priority robot plans its path considering the former as a dynamic obstacle and broadcast, and the process continues for robots of the next priorities. After all robots compute their plans in this decentralized *manner, they leave their parking cells and follow their pre-decided plan of movement. If new robots are added during the movement of these robots, the online algorithm can admit them in the movements, but cannot ensure that all other robots who have a lower priority than the newly entered one will update their path plans accordingly. Thereby, this online version of the PP algorithm cannot ensure the priorities among all the participants which is guaranteed by the offline PP algorithm if all robots started simultaneously. Hence, this scheme is not robust against entry-exit.*

In the following section, we show that our proposed mechanism satisfies all these properties. In Section 6, we consider a real warehouse setting and exhibit the performance of **OMCoRP** in practice.

5 Theoretical Guarantees

The property of truthfulness is important in the multi-agent setting since it ensures that the allocation decision is taken on the *true* values of $v_i(t)$'s and the actual locally efficient allocations were done.

Theorem 1 **OMCoRP** *is dominant strategy truthful.*

Proof: This proof is a standard exercise in the line of the proof for Vickery-Clarke-Groves (VCG) mechanism (Vickrey 1961; Clarke 1971; Groves 1973). **OMCoRP** follows the VCG allocation and payment locally at every intersection calculated by the LIM. Hence, the payoff of robot i at intersection k is given by (for brevity of notation, we hide the time argument in every function and write a_k^* as $a_k^*(v_i, v_{-i})$)

$$\begin{aligned}&\mathbf{val}_i(a_k^*(v_i, \hat{v}_{-i}), v_i) - p_i(v_i, \hat{v}_{-i}) \\ &= \sum_{j \in N_k} \mathbf{val}_j(a_k^*(v_i, \hat{v}_{-i}), \hat{v}_j) - \sum_{j \in N_k \setminus \{i\}} \mathbf{val}_j(a_k^{*, N_k \setminus \{i\}}, \hat{v}_j) \\ &\geqslant \sum_{j \in N_k} \mathbf{val}_j(a_k^*(\hat{v}_i, \hat{v}_{-i}), \hat{v}_j) - \sum_{j \in N_k \setminus \{i\}} \mathbf{val}_j(a_k^{*, N_k \setminus \{i\}}, \hat{v}_j) \\ &= \mathbf{val}_i(a_k^*(\hat{v}_i, \hat{v}_{-i}), v_i) - p_i(\hat{v}_i, \hat{v}_{-i}).\end{aligned}$$

The first equality is obtained by writing and reorganizing the expression for p_i. The inequality holds since by definition $\sum_{j \in N_k} \mathbf{val}_j(a_k^*(v_i, \hat{v}_{-i}), \hat{v}_j) \geqslant \sum_{j \in N_k} \mathbf{val}_j(a_k, \hat{v}_j)$ for every a_k; in particular, we chose $a_k^*(\hat{v}_i, \hat{v}_{-i})$. The last equality is obtained by reorganizing the expressions again. ■

OMCoRP redistributes the generated money from a particular intersection k to the robots who are not part of it at that time step (see Step 5 of Algorithm 1). Clearly, this does not affect the truthfulness properties. The generated surplus before redistributing can be shown to be always non-negative.[10] The allocation of **OMCoRP**, given by

[9]We use the subscript $-i$ to denote all the agents except agent i, therefore, $v_{-i} := (v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n)$.

[10]The intuition for this claim is that the absence of a robot re-

Equation (2), maximizes the sum-value at every intersection. Therefore, it is locally efficient. Hence we get the following theorem.

Theorem 2 **OMCoRP** *is budget balanced and locally efficient.*

OMCoRP satisfies certain properties by construction. The following claim summarizes them and we explain it below.

Claim 1 **OMCoRP** *is collision-free, locally deadlock-free, and robust against entry or exit.*

In **OMCoRP**, each robot near an intersection point does message exchange with the LIM, and then move synchronously according to the recommendation of the LIM. The LIM computes the allocation which keeps one cell in the intersection empty (by definition of the allocation set A_k). Thus the robots avoid collision and deadlock.

Note that **OMCoRP** depends only on the values reported by the robots that are already in an intersection or are about to enter it. A newly entered robot can at most take part in such an interaction, but cannot change the way other robots in other intersections interact with each other. Hence, the properties that those robots were satisfying, e.g., truthfulness, budget balance, efficiency etc., before the entry of this robot continue to be satisfied. Hence we get the claim.

In the following section, we investigate the performance of **OMCoRP** in real-world scenarios.

6 Experiments

While **OMCoRP** satisfies several desirable properties of a collision avoidance mechanism, its scalability, time complexity to find a collision-avoiding path, and differentiated treatment with different classes of robots are not theoretically captured. This is why an experimental study is called for. Note that the mechanisms that fall in the third set of approaches in §1 either only use the bidding part of the auctions and not the payment which is essential to guarantee truthfulness in an auction (Lagoudakis et al. 2005; Bererton, Gordon, and Thrun 2004; Nunes and Gini 2015; Calliess, Lyons, and Hanebeck 2011) or is a reduced combinatorial auction of a centralized offline algorithm (Amir, Sharon, and Stern 2015). In either way they are incomparable with **OMCoRP**, which is truthful and online. We compare **OMCoRP** with two widely used offline path planning algorithms for multi-agent path finding – (a) M* (Wagner and Choset 2011): optimal, but has a significant time complexity, and (b) prioritized planning (PP) (van den Berg and Overmars 2005): suboptimal, but has low time complexity.

To evaluate **OMCoRP** experimentally, we use a 2-D rectangular workspace representing a track network. An example of such a workspace of size 16×16 is shown in Fig. 3. A robot picks and delivers an object from and to a cell in the service area. Each robot follows the traffic rules in the road network, and moves along with the directions of the arrows from the source to the destination. Note that this network can be represented as a grid graph with edges in both directions

duces congestion, which leads to the other robots being allowed to move. This increases the sum of the values of the other robots in the absence of that robot. We skip a formal proof of this fact.

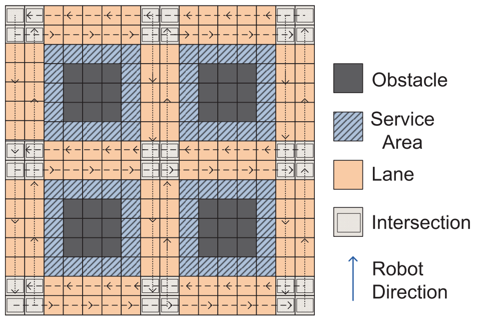


Figure 3: Illustration of a 16×16 workspace.

over the grid. It can change its direction only within an intersection cell, but its options are restricted by the available traffic directions in that particular intersection cell.

For the experiments, time is slotted and we assume that the robots move synchronously according to that slotted time system. We use two measures to evaluate the performance of **OMCoRP**: (a) *makespan*, which is the *maximum* number of time steps to reach the destination from the source for all the robots, and (b) *total cost*, which is the *sum* of the number of time steps for all the robots to reach their destinations.

A robot is generated uniformly at random from one of the three classes: *economy, regular*, and *premium* having weights 0.02, 0.065, and 0.2 respectively[11]. At every time step, the true *valuation* of a robot entering an intersection is assumed to be $(t_{\text{wait}} + 1) \times w$, where t_{wait} is the wait time of the robot till that instant and w is the weight as described above. All the robots start simultaneously in all the experiments except the one in Section 6.4, where arrivals are dynamic.

We have implemented **OMCoRP** in Python. The source code is available at https://github.com/iitkcpslab/OMCoRP. The simulations have been performed in a 64-bit Ubuntu 14.04 LTS machine with Intel(R) Core TM i7-4770 CPU @3.40 GHz $\times$ 20 processors and 128 GB RAM. Each run for a specific number of robots and a workspace size is performed 20 times to calculate the average of the result. The source and goal locations of the robots are selected independently and uniformly at random from the cells in the service area.

6.1 Scalability

We evaluated the scalability of **OMCoRP** on workspaces of four different sizes: 100×100, 198×198, 401×401 and 499×499[12], and for different number of robots between 10 and 500. Fig. 4 shows how the computation time of **OMCoRP** varies with the number of robots and the size of the workspace. The computation time of **OMCoRP** is the sum of two components: (a) the time for an *offline* path computation and (b) the time required to run the *online* mechanism at different LIMs (given by Algorithm 1).

In our experiments, we assume that the robots compute

[11]Weights are increasing with a rough multiplying factor of 3.25.

[12]These numbers ensure a regular pattern of the workspace of Fig. 3.

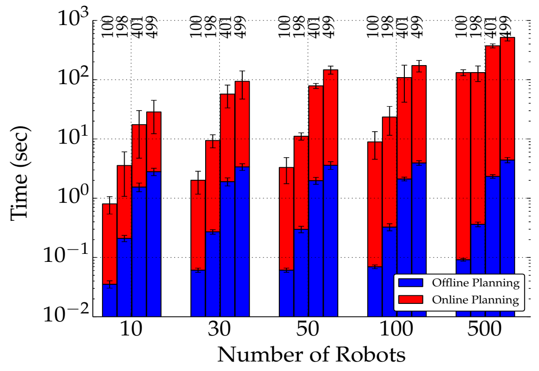


Figure 4: Offline and online components of the computation time of **OMCoRP**. The numbers on the bars show the workspace width for each number of robots.

their paths to reach the destination using A* algorithm (Hart, Nilsson, and Raphael 1968). As this computation can take place in parallel, we count the *maximum* of the path computation times for all the robots as the offline computation time (part (a)). The average computation time is insensitive to the number of robots for a fixed workspace (the blue bar in Fig. 4 is roughly the same).

For part (b), we assume that each time step begins with a *tiny duration* when the online component of the mechanism executes at an intersection. We assume that the remaining time in a time step is sufficient for a robot to move into its intended cell. We take the product of this tiny duration with the makespan to find the total time spent in part (b). The tiny duration has been decided based on extensive simulation with different reasonably-sized workspaces and robot populations.

The online part of the computation actually executes the allocation and payment mechanism whenever there is a collision situation. This part of the mechanism takes some small time to decide which robots should move before the robots can actually start moving. But since all robots must move in a synchronized manner, this duration within which the online part of the planning is executed has to be the same for all the robots. So, we set this 'tiny duration', which can be experimentally found for a given workspace size and number of robots, to be the maximum time over all runs of this experiment and reserve this time in the beginning of every time-slot within which the decision of which robots will move is guaranteed to be completed. Since this duration is an overhead for every time-slot, we consider the entire time spent in this overhead as the online planning time. The red bar in Fig. 4 represents the average of this total overhead (= makespan $\times$ the tiny duration).

6.2 Comparison With Static Multi-Robot Planning Algorithms

We compare the performance of **OMCoRP** with that of M* (Wagner and Choset 2011) and prioritized planning (PP) (van den Berg and Overmars 2005), two state-of-the-art multi-robot path planning algorithms. The goal of these experiments is to show that though **OMCoRP** has been de-

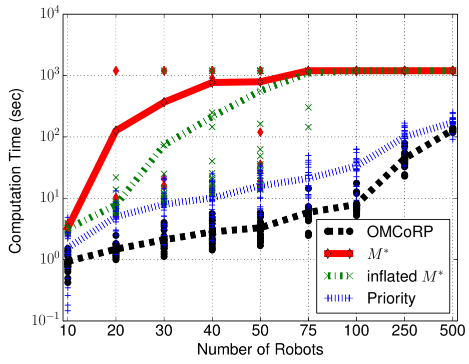


Figure 5: The y-axis shows the total computation time (in sec) of the different algorithms and the x-axis shows the number of robots. Workspace size 100×100.

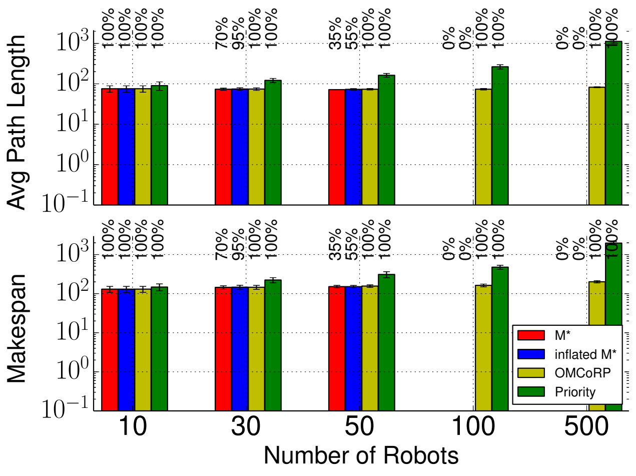


Figure 6: Average path length and makespan for different mechanisms. The numbers on the bar denote the percentage of cases where the mechanism could find a path within the timeout period.

signed for competitive robots, it is also effective as an online planning mechanism for a *cooperative* multi-robot system. In that case, we can omit the payment component of **OMCoRP** as we do not need to ensure the truthfulness of the participating robots. The original version of M* ensures the optimality of the generated multi-robot plan in terms of the total cost. However, a variant of M*, called *inflated* M*, can produce a sub-optimal plan faster than the original M*. **OMCoRP** is also compared with inflated M*. The PP algorithm also does not provide any optimality guarantee, but can generate collision-free paths much more time-efficiently than both versions of M*.

We compare **OMCoRP**, M*, inflated M*, and PP for a 100×100 workspace and different number of robots up to 500 robots. We have set a timeout of 1200s (20min). If the computation does not finish before the timeout, we consider the timeout duration as the computation time of the algorithms (which is a lower bound). The experimental results are shown in Fig. 5. The lines connect through the average values of the computation times, while the actual computation times for 20 runs for every number of robots are scat-

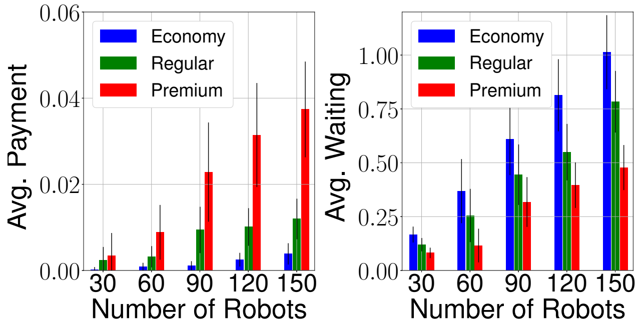


Figure 7: Average waiting time (in sec) and payments under **OMCoRP** for different classes of robots.

tered according to their values in the figure. The results show that **OMCoRP** outperforms all the other three algorithms in terms of computation time. M* and inflated M* algorithms do not scale beyond 75 robots for this timeout.

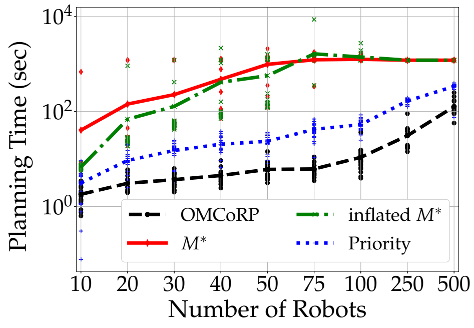
Fig. 6 compares the makespan (maximum time among all robots to reach destination) and average path execution time (the average of the individual path lengths) for the mechanisms for a workspace of size 100×100. The method of the plot is identical to that of Fig. 5. The numbers on top of the bars show the percentage of successful completion (not hitting timeout). The results show that there is not much difference in the two metrics of these mechanisms (except PP, which is statistically significantly larger in both metrics) and that **OMCoRP** provides a path which is very close in length to that of the optimal path generated by M*.

6.3 Delays of Different Priority Classes

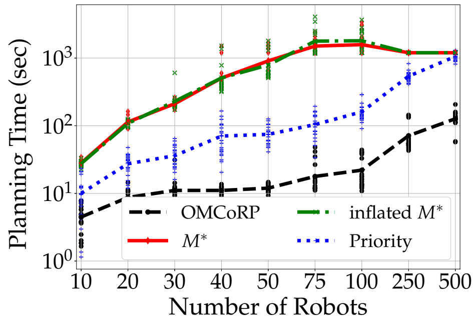
We compare the waiting time and payments for the robots of different classes (economy, regular, and premium) under **OMCoRP**. The first and second subplots of Fig. 7 show the average payments and waiting times respectively of the different classes of robots for a 100×100 workspace. We see that when the number of robots is large, **OMCoRP** prioritizes the higher classes for a greater payment. The difference in the waiting times in such cases is statistically significant.

6.4 Capability of Handling Dynamic Robot Arrival

In this section, we study the robustness of **OMCoRP** against dynamic robot arrivals. The setup remains similar to the previous subsections with the following differences. We partition the total number of robots into two groups of equal size for this evaluation. The first group of robots arrive at the beginning. The rest 50% of the robots arrive independently and uniformly at random within the time interval of zero and the length of the workspace. In **OMCoRP**, a newly arrived robot computes its own path by using the A* algorithm (Hart, Nilsson, and Raphael 1968), and starts to follow that path immediately. However, in M*, a newly arrived robot requests path to the centralized M* path planner. The M* planner collects such requests and waits for re-planning until the number of the newly arrived robots exceeds a predefined threshold. In our experiments, this threshold is set to 2. During the re-planning, the M* planner considers the current



(a) Workspace size 100×100



(b) Workspace size 198×198

Figure 8: The y-axis shows the total computation time (in sec) of the algorithms and the x-axis shows the number of robots.

locations of the previously arrived robots as their source locations and excludes the robots which already have reached their respective goal locations. For PP, the robots that arrive later are considered to have a lower priority than the robots arrived already, and their paths are computed considering the previous robots' positions as dynamic obstacles. Fig. 8 shows the results of the experiments for two different workspace sizes: 100×100 and 198×198.

6.5 Experiments With ROS

We have simulated **OMCoRP** for up to 10 TurtleBots on ROS (Quigley et al. 2009). We have used a workspace of size 14×14 (similar to Fig. 3) with each grid cell of length 1m. The linear and the angular velocity of the TurtleBots have been chosen in such a way that each motion primitive takes 6.5s for execution. The maximum time for the online component of the computation time (decides the length of the tiny duration) observed in all our experiments is about 60ms. The video of our experiments is available at: https://youtu.be/9JRddBuVHoA.

7 Summary and Future Work

We presented a scalable online collision avoidance mechanism for a *competitive* multi-robot system which satisfies several desirable properties. In future, we would like to extend the algorithm to more than two dimensions and conduct experiments on a real multi-robot system.

References

Amir, O.; Sharon, G.; and Stern, R. 2015. Multi-Agent Pathfinding as a Combinatorial Auction. In *AAAI*, 2003–2009.

Azarm, K.; and Schmidt, G. 1997. Conflict-free motion of multiple mobile robots based on decentralized motion planning and negotiation. In *ICRA*, volume 4, 3526–3533.

Bererton, C.; Gordon, G. J.; and Thrun, S. 2004. Auction mechanism design for multi-robot coordination. In *NIPS*, 879–886.

Bogue, R. 2016. Growth in e-commerce boosts innovation in the warehouse robot market. *Industrial Robot: An International Journal* 43(6): 583–587.

Börgers, T. 2015. *An introduction to the theory of mechanism design*. Oxford University Press, USA.

Calliess, J.-P.; Lyons, D.; and Hanebeck, U. D. 2011. Lazy auctions for multi-robot collision avoidance and motion control under uncertainty. In *AAMAS*, 295–312.

Chen, Y. F.; Liu, M.; Everett, M.; and How, J. P. 2017. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In *ICRA*, 285–292.

Chun, L.; Zheng, Z.; and Chang, W. 1999. A decentralized approach to the conflict-free motion planning for multiple mobile robots. In *ICRA*, volume 2, 1544–1549.

Clarke, E. H. 1971. Multipart Pricing of Public Goods. *Public Choice* 11: 17–33.

Desai, A.; Saha, I.; Yang, J.; Qadeer, S.; and Seshia, S. A. 2017. DRONA: a framework for safe distributed mobile robotics. In *ICCPS*, 239–248.

Desaraju, V.; and How, J. P. 2012. Decentralized path planning for multi-agent teams with complex constraints. *Autonomous Robots* 32(4): 385–403.

Dhinakaran, A.; Chen, M.; Chou, G.; Shih, J. C.; and Tomlin, C. J. 2017. A hybrid framework for multi-vehicle collision avoidance. In *CDC*, 2979–2984.

Erdmann, M.; and Lozano-Perez, T. 1986. On multiple moving objects. In *ICRA*, volume 3, 1419–1424.

Gavran, I.; Majumdar, R.; and Saha, I. 2017. Antlab: A Multi-Robot Task Server. *ACM Trans. Embedded Comput. Syst.* 16(5): 190:1–190:19.

Gibbard, A. 1973. Manipulation of Voting Schemes. *Econometrica* 41: 587–602.

Groves, T. 1973. Incentives in Teams. *Econometrica* 41: 617–631.

Guizzo, E. 2008. Three Engineers, Hundreds of Robots, One Warehouse. *IEEE Spectrum* 45(7): 26–34.

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.* 4(2): 100–107.

Hennes, D.; Claes, D.; Meeussen, W.; and Tuyls, K. 2012. Multi-robot collision avoidance with localization uncertainty. In *AAMAS*, 147–154.

Hoffmann, G.; and Tomlin, C. 2008. Decentralized cooperative collision avoidance for acceleration constrained vehicles. In *CDC*, 4357–4363.

Jager, M.; and Nebel, B. 2001. Decentralized collision avoidance, deadlock detection, and deadlock resolution for multiple mobile robots. In *IROS*, volume 3, 1213–1219.

Lagoudakis, M. G.; Markakis, E.; Kempe, D.; Keskinocak, P.; Kleywegt, A. J.; Koenig, S.; Tovey, C. A.; Meyerson, A.; and Jain, S. 2005. Auction-Based Multi-Robot Routing. In *RSS*, volume 5, 343–350.

Nunes, E.; and Gini, M. 2015. Multi-robot auctions for allocation of tasks with temporal constraints. In *AAAI*, 2110–2116.

Olfati-Saber, R.; Fax, J.; and Murray, R. 2007. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE* 95(1): 215–233.

Pallottino, L.; Scordio, V.; and Bicchi, A. 2004. Decentralized cooperative conflict resolution among multiple autonomous mobile agents. In *CDC*, volume 5, 4758–4763.

Purwin, O.; D'Andrea, R.; and Lee, J. 2008. Theory and implementation of path planning by negotiation for decentralized agents. *Robotics and Autonomous Systems* 56(5): 422–436.

Quigley, M.; Gerkey, B.; Conley, K.; Faust, J.; Foote, T.; Leibs, J.; Berger, E.; Wheeler, R.; and Ng, A. Y. 2009. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*.

Roberts, K. 1979. *The Characterization of Implementable Choice Rules*, chapter Aggregation and Revelation of Preferences, 321–348. North Holland Publishing.

Saha, I.; Ramaithitima, R.; Kumar, V.; Pappas, G. J.; and Seshia, S. A. 2014. Automated composition of motion primitives for multi-robot systems from safe LTL specifications. In *IROS*, 1525–1532.

Saha, I.; Ramaithitima, R.; Kumar, V.; Pappas, G. J.; and Seshia, S. A. 2016. Implan: Scalable Incremental Motion Planning for Multi-Robot Systems. In *ICCPS*, 43:1–43:10.

Satterthwaite, M. 1975. Strategy-Proofness and Arrow's Conditions: Existence and Correspondence Theorems for Voting Procedures and Social Welfare Functions. *Journal of Economic Theory* 10: 187–217.

Shoham, Y.; and Leyton-Brown, K. 2008. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press.

Snape, J.; Van Den Berg, J.; Guy, S. J.; and Manocha, D. 2010. Smooth and collision-free navigation for multiple robots under differential-drive constraints. In *IROS*, 4584–4589.

Takei, R.; Huang, H.; Ding, J.; and Tomlin, C. J. 2012. Time-optimal multi-stage motion planning with guaranteed collision avoidance via an open-loop game formulation. In *ICRA*, 323–329.

Turpin, M.; Michael, N.; and Kumar, V. 2014. Capt: Concurrent assignment and planning of trajectories for multiple robots. *Int. J. Robotics Res.* 33(1): 98–112.

van den Berg, J.; and Overmars, M. 2005. Prioritized motion planning for multiple robots. In *IROS*, 2217–2222.

Velagapudi, P.; Sycara, K.; and Scerri, P. 2010. Decentralized prioritized planning in large multirobot teams. In *IROS*, 4603–4609.

Vickrey, W. 1961. Counter Speculation, Auctions, and Competitive Sealed Tenders. *Journal of Finance* 16(1): 8–37.

Wagner, G.; and Choset, H. 2011. M*: A Complete Multirobot Path Planning Algorithm with Performance Bounds. In *IROS*, 3260–3267.

Wulfraat, M. 2016. Is Kiva Systems a Good Fit for Your Distribution Center? An Unbiased Distribution Consultant Evaluation. http://www.mwpvl.com/html/kiva_systems.html. Accessed: October 2020.

Yu, J.; and LaValle, S. M. 2013. Planning optimal paths for multiple robots on graphs. In *ICRA*, 3612–3617.