Prioritized Planning Algorithms for Trajectory Coordination of Multiple Mobile Robots

Michal Čáp, Peter Novák, Alexander Kleiner, Martin Selecký

***Abstract*—An important capability of autonomous multi-robot systems is to prevent collision among the individual robots. One approach to this problem is to plan conflict-free trajectories and let each of the robots follow its pre-planned trajectory. A widely used practical method for multi-robot trajectory planning is prioritized planning, which has been shown to be effective in practice, but is in general incomplete. Formal analysis of instances that are provably solvable by prioritized planning is still missing. Moreover, prioritized planning is a centralized algorithm, which may be in many situations undesirable.**

In this paper we a) propose a revised version of prioritized planning and characterize the class of instances that are provably solvable by the algorithm and b) propose an asynchronous decentralized variant of prioritized planning, which maintains the desirable properties of the centralized version and in the same time exploits the distributed computational power of the individual robots, which in most situations allows to find the joint trajectories faster.

The experimental evaluation performed on real-world indoor maps shows that a) the revised version of prioritized planning reliably solves a wide class of instances on which both classical prioritized planning and popular reactive technique ORCA fail and b) the asynchronous decentralized algorithm provides solution faster than the previously proposed synchronized decentralized algorithm.

***Note to Practitioners*—Consider a large warehouse in which the goods are stored and retrieved by autonomous transport robots. One option is to ignore interaction between the vehicles during the route planning for each robot and handle the conflicts only during the route execution. However, such approach is prone to deadlocks, i.e. to a situations during which some of the robots mutually block each other, cannot proceed and fail to complete their transportation task. An alternative approach would involve planning collision-free routes for each robot before the robots start executing them. However, the general methods for this problem that guarantee a solution are impractical due to their high computational complexity. In this paper, we show that a simple prioritized approach in which robots plan their routes one after another is guaranteed to find collision-free trajectories for a well-defined class of practical problems. In particular, if the systems resembles human-made transport infrastructures by requiring that the start and destination position of each vehicle must never obstruct other vehicles from moving, then the proposed approach is guaranteed to provide a solution. E.g., in such a warehouse application, the collision-free routes can efficiently computed by the prioritized approach. The paper contains formal condition that characterizes the problem instances for which is the method guaranteed to work.**

Further, we propose a new decentralized adaptation of the prioritized algorithm that can be used in multi-robot systems without a central solver. This technique can be used to find coordinate trajectories just by running simple negotiation protocol between the individual robots. The paper contains analysis showing that the decentralized algorithm exhibits desirable theoretical properties and experimental comparison of performance of different variations of centralized and decentralized algorithms.

***Index Terms*—multi-robot systems, trajectory planning, collision avoidance, decentralized algorithms**

I. Introduction

When mobile robots autonomously operate in a shared space, one of the crucial issues is how to coordinate the trajectories of individual robots so as to prevent potential collisions. The two most commonly used classes of methods that deal with this problems are multi-robot planning and reactive collision avoidance.

Multi-robot motion planners take into consideration the start and goal position of each robot and plan coordinated trajectories that are mutually conflict-free. If the robots execute the resulting joint plan precisely (or within some given tolerance), it is guaranteed that the robots will reach their goals without collision. However, it is known that even the simplest variants of multi-robot path planning problem are intractable. Deciding whether a coordinated collision-free paths exist for multiple discs moving amidst polygonal obstacles is known to be strongly NP-hard [8]; the same task involving rectangular objects in an empty room is known to be in PSPACE-hard [4].

The multi-robot planners are typically based either on the coupled heuristic search in the joint state space of all robots or on decoupled planning. The coupled approaches typically find optimal solutions [9], [10], [15], but do not scale well with the increasing number of conflicting robots.

On the other hand, decoupled approaches plan independently for each robot. They can be fast enough for real-time applications, but they typically suffer from incompleteness.

A widely used decoupled scheme for the multi-robot motion planning that has been shown to be effective in practice is prioritized planning [3]. In prioritized planning, each robot is assigned a unique priority and the algorithm proceeds sequentially from the highest priority robot to the lowest priority one. At each iteration, one of the robots plans its trajectory such that it avoids the higher-priority robots. Such a greedy approach is clearly incomplete if we allow arbitrary maps and arbitrary start and goal locations for each robot, but in relatively sparse environments, the techniques tends to perform well.

Recently, Velagapudi et al. presented a decentralized version of prioritized planning technique for teams of mobile robots [14], which is able to utilize the distributed computational resources to reduce the time needed to find a solution. Since the algorithm proceeds in globally-synchronized rounds, faster-computing robots have to wait at the end of each round for the longest-computing robot and thus the distributed computational power may not be used efficiently.

The contribution of this paper is twofold: 1) We propose a revised version of the prioritized planning scheme and show that for this revised version it is possible to provide sufficient conditions under which is the algorithm guaranteed to provide a solution. 2) We propose a novel asynchronous decentralized variant of both classical and revised prioritized planning scheme that is guaranteed to terminate and inherits completeness properties from the respective centralized counterpart. We experimentally show that asynchronous decentralized algorithm exhibits better utilization of the distributed computational resources and thus provides faster convergence times compared to the previously presented synchronized approach.

Partial results of the presented work appeared in [2], [13], where the focus was on the design of asynchronous version of decentralized prioritized planning. Here, we extend our previous work by proposing the revised version of prioritized planning scheme, by theoretical analysis of the properties of all discussed algorithms, by performing experimental comparison on real-world indoor maps and by including reactive techniques into the comparison.

II. Problem Definition

Consider n circular robots operating in a 2-d workspace $\mathcal{W} \subseteq \mathbb{R}^2$. The subset of $\mathcal{W}$ occupied by the body of robot i when its center is on position x is denoted as $R_i(x)$. The maximum speed the robot i can move at is denoted as v_i. Each robot is assumed to be assigned a *task* that involves moving from its start position s_i to some goal position g_i and stay there. We assume that the start and goal positions of all robots are mutually disjunct, i.e. the bodies of robots do not overlap when the robots are on their start positions and when they are on their goal positions.

A path $p : [0,1] \to \mathcal{W}$ of robot i in workspace $\mathcal{W}$ is called *satisfying* if it starts at the robot's start position s_i, ends at robot's goal position g_i, and the body of robot whose center follows the path p always lies entirely in $\mathcal{W}$. A trajectory $\pi : [0,\infty) \to \mathcal{W}$ is a mapping from time points to positions in workspace and unlike a path it carries information about how it should be executed in time. Analogically, a trajectory of robot i is called *satisfying* if it starts at the robot's start position s_i, finally reaches and stays at the goal position g_i, the body of robot i whose center follows the trajectory π always lies entirely in $\mathcal{W}$, and the robot never moves faster than its maximum speed v_i.

The trajectories π_i, π_j of two robots i, j are said to be *conflict-free* if the bodies of the robots i, j never intersect when they follow the trajectories π_i and π_j.

Problem 1 (Trajectory Coordination Problem)**.** Given a workspace $\mathcal{W}$ and tasks $\langle s_1, g_1\rangle, \ldots, \langle s_n, g_n\rangle$ for robots $1, \ldots, n$, find trajectories $\pi_1, \ldots, \pi_n$ such that each trajectory π_i is satisfying for robot i and trajectories π_i, π_j of every two different robots i, j are mutually conflict-free.

Notation

The following shorthand notations will be used to talk about regions occupied by a different subsets of robots at their start and goal positions:

$$\begin{array}{rclcrcl} S^i & := & R_i(s_i) & & G^i & := & R_i(g_i) \\ S^{>i} & := & \bigcup\limits_{j=i+1,\ldots,n} R_j(s_j) & & G^{<i} & := & \bigcup\limits_{j=1,\ldots,i-1} R_j(g_j) \\ S & := & \bigcup\limits_{j=1,\ldots,n} R_j(s_j) & & G & := & \bigcup\limits_{j=1,\ldots,n} R_j(g_j) \end{array}$$

Further, we will work with the concept of a space-time region: When a spatial object, such as the body of a robot, follows a given trajectory, then it can be thought of as occupying a certain region in space-time $\mathcal{T} := \mathcal{W} \times [0,\infty)$. A dynamic obstacle Δ is then a region in such a space-time $\mathcal{T}$. If $(x, y, t) \in \Delta$, then we know that the spatial position (x, y) is occupied by dynamic obstacle Δ at time t. The function

$$R_i^{\Delta}(\pi) := \{(x, y, t) : t \in [0,\infty) \wedge (x, y) \in R_i(\pi(t))\}$$

maps trajectories of a robot i to regions of space-time that the robot i occupies when its center point follows given trajectory π. As a special case, let $R_i^{\Delta}(\emptyset) := \emptyset$.

Assumptions on Communication

We assume that each robot is equipped with an independent computation unit and a wireless device for communication with other robots. Wireless communication channels are typically implemented as broadcast channels, where each communicated message is broadcast, but ignored by the nodes that are not among the declared recipients of the message. In such a channel a single broadcast message uses the same channel capacity as a single point-to-point message and thus we will prefer to perform a single broadcast instead of sending several point-to-point messages. Further, in the following discussion we will assume that such a communication channel is reliable, i.e. each broadcast messages is eventually received by all robots in the system, and that the communication channel preserves the ordering of messages that were sent in.

III. Prioritized Planning

A straightforward approach to solve the trajectory coordination problem would be to see all robots in the system as one composite robot with many degrees of freedom and use some path planning algorithm to find a joint path for all the robots. However, the size of such a joint configuration space is exponential in the number of robots and thus this approach quickly becomes impractical if one wants to plan for more than a few robots. A pragmatic approach that is often useful even for large multi-robot teams is prioritized planning. The idea has been first articulated by Erdman and Lozano-Pérez in [3]. Other works such as [11], [1] investigate techniques for choosing a good prioritization for the robots.

Classical Prioritized Planning

In prioritized planning each robot is assigned a unique priority. The trajectories for individual robots are then planned sequentially from the highest priority robot to the lowest priority one. For each robot a trajectory is planned that avoids both the static obstacles in the environment and the higher-priority robots moving along the trajectories planned in the

Algorithm 1: Classical Prioritized Planning

```
Algorithm PP
    Δ ← ∅;
    for i ← 1 . . . n do
        π_i ← Best-traj(W, Δ);
        if π_i = ∅ then
            report failure and terminate
        Δ ← Δ ∪ R_i^Δ(π_i);
Function Best-traj(W', Δ)
    return optimal satisfying trajectory for robot i in W'
    that avoids regions Δ if it exists, otherwise return ∅
```

previous iterations. The pseudocode of classical prioritized planning is in Algorithm 1.

The algorithm iterates over the robots, starting from the highest-priority robot 1 to the lowest-priority robot n. During i-th iteration the algorithm computes a trajectory for robot i that avoids the space-time regions occupied by robots $1, \ldots, i-1$.

The trajectory of robot i is computed in `Best-traj`$(\mathcal{W}', \Delta)$ function. The function returns a trajectory for robot i such that body of the robot always stays inside the static workspace $\mathcal{W}'$ and avoids dynamic regions Δ occupied by other robots. Such a function would be in practice implemented using some application-specific technique for motion planning with dynamic obstacles, e.g. [12] or [6]. As it will become clear later, it is desirable that this function is implemented using an algorithm that offers some form of completeness, since this property will be inherited also by the multi-robot algorithm.

Properties: The algorithm terminates either with success or with failure in at most n iterations. The successful termination occurs in exactly n iterations if valid trajectories for all robots have been found. The termination with failure occurs if there exists a robot for whom no satisfying trajectory that avoids higher-priority robots have been found.

When the algorithm terminates successfully, each robot is assigned a trajectory that is conflict-free with the trajectories of all other robots. This follows from the fact that the final trajectory of each robot i is conflict-free with the higher-priority robots, since robot i avoided collision with them and also with lower-priority robots since they avoided conflict with the trajectory of robot i themselves.

Prioritized planning is in general incomplete, consider the counter-example [7] depicted in Figure 1:

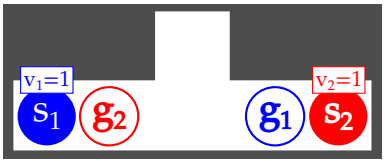


Figure 1: The picture shows two robots desiring to move from s_1 to g_1 (s_2 to g_2 resp.) in a corridor that is only slightly wider than a body of a single robot. The scenario assumes that both robots have identical maximum speeds. Observe that irrespective of which robot starts planning first, its trajectory will be in conflict with all satisfying trajectories of the robot that plans second.

Let us now analyze when is prioritized planning bound to fail. The algorithm fails to find a trajectory for robot i if 1) no satisfying path exists for robot i, i.e. the robot cannot reach its destination even if there are no other robots in the workspace; 2) every satisfying trajectory of robot i is in conflict with some higher-priority robot. There are two types of conflicts that can occur between a satisfying trajectory π of robot i and a higher-priority robot:

Type A: Occurs if trajectory π is in conflict with a higher-priority robot who has reached and is "sitting" at its destination, i.e. it is blocked by a static higher-priority robot. The following Figure shows a scenario where all satisfying trajectories of a robot are in Type A conflict:

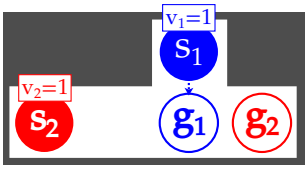


Figure 2: Robot 1 travels from s_1 to g_1, robot 2 travels from s_2 to g_2. Both robots have identical maximum speed $v = 1$. Robot 1 plans first and adopts a straight line trajectory from s_1 to g_1 at the maximum speed, because it ignores the task of robot 2. Consequently all satisfying trajectories of robot 2 will be in Type A conflict with robot 1.

Type B: Occurs if trajectory π of robot i is in conflict with a higher-priority robot who is moving towards its destination, i.e. it is "run over" by a moving higher-priority robot. The following Figure shows a scenario where all satisfying trajectories of a robot are in Type B conflict:

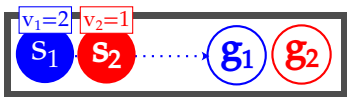


Figure 3: Robot 1 travels from s_1 to g_1, robot 2 travels from s_2 to g_2. Assume that robot 1 can travel *twice as fast* as robot 2. Then, robot 1 will plan first and adopts a straight line trajectory from s_1 to g_1 at the maximum speed, because it ignores the task of robot 2. Robot 2 plans second, but all satisfying trajectories for robot 2 are in Type B conflict with robot 1. If no satisfying trajectory exists or all satisfying trajectories are engaged in a Type A or Type B conflict, then prioritized planing fails to find a satisfying and consistent trajectory for robot i and terminates with failure.

A question that naturally arises is whether it would be possible to restrict the class of solvable instances or to alter the prioritized planning algorithm such that there will always be at least one trajectory without neither Type A nor Type B conflict for each robot.

One way to ensure that there will be a satisfying trajectory without Type A conflict for every robot is to only consider instances, where each robot has a path to its goal that avoids goal regions of all higher-priority robots. When each robot follows such a path, then they cannot be engaged in a Type A conflict, because a Type A conflict can only occur at the goal region of one of the higher-priority robots.

Unfortunately, the existence of a trajectory without Type B conflict is difficult to guarantee in classical prioritized planning, since higher-priority robots completely ignore interactions with lower-priority robots when planning their trajectories. To ensure that each robot will have a satisfying trajectory without Type B conflict, all higher-priority robots would have to plan their trajectories so that the lower-priority robots are always left with some alternative trajectory that can be used to avoid the potential conflicts of this type.

One way to ensure that there will be a satisfying trajectory without Type B conflict for every robot is to consider only instances where each robot has a path to its goal that avoids start region of lower-priority robots and enforce that the trajectory of each robot will avoid start regions of all lower-priority robots. When this is ensured, then any robot will always have a fall-back option to wait at its start position (since no higher-priority robot can run over its start region) until its desired path is clear of all higher-priority robots. Thus it can always avoid Type B conflicts.

Moreover, if the robot continues by following a path that avoids goal regions of higher-priority robots, then the resulting trajectory is also guaranteed to avoid the Type A conflicts.

Theorem 2. *Let us have a trajectory coordination problem with workspace $\mathcal{W}$ and tasks $\langle s_1, g_1\rangle, \ldots, \langle s_n, g_n\rangle$ for robots $1, \ldots, n$. If for every robot i there exist a $S^{>i}$-avoiding and $G^{<i}$-avoiding satisfying path, then a sequential conflict-free solution can be constructed.*

Proof: We will construct the solution inductively as follows:

Induction assumption: Trajectories of robots $1, \ldots, i-1$ are satisfying and $S^{>i-1}$-avoiding.

Base step (robot 1): Robot 1 is the highest-priority robot. There are no higher-priority robots that the robot 1 needs to avoid. From our assumption there exists a path p that is satisfying for robot 1 and $S^{>1}$-avoiding. A satisfying and $S^{>1}$-avoiding trajectory for robot 1 can be simply constructed by following the path p at an arbitrary positive speed. Such a trajectory can always be constructed, therefore the algorithm will not report failure when planning for robot 1.

Induction step (robot i): From our assumption there exists a path p that is satisfying for robot i, $S^{>i}$-avoiding and $G^{<i}$-avoiding. Since all trajectories for robots $1, \ldots, i-1$ are satisfying (i.e. eventually reach the goal and stay there) then there must exists a time point $\bar{t}$ after which all robots $1, \ldots, i-1$ have reached and will stay at their goal. A satisfying and $S^{>i}$-avoiding trajectory for robot i that is conflict-free with all robots $1, \ldots i-1$ can be constructed as follows:

- In interval $[0, \bar{t}]$ stay at s_i. The trajectory cannot be in conflict with higher-priority robots during this interval, because all trajectories of robots $1, \ldots, i-1$ are $S^{>i-1}$ and thus also S^{i}-avoiding.
- In interval $[\bar{t}, \infty]$ follow path p until the goal position g_i is reached. The path p avoids regions $G^{<i}$ and thus the trajectory cannot be in collision with any of the higher-priority robots $1, \ldots, i-1$ because they are at their goal positions during this time interval, which the path p avoids.

Such a trajectory can always be constructed.

The trajectories of robots $1, \ldots, i-1$ are satisfying and $S^{>i-1}$-avoiding, which implies that they are also $S^{>i}$-avoiding. The newly computed trajectory for robot i is satisfying and $S^{>i}$-avoiding. By taking the union of the old set of trajectories and the new trajectory we have a set of trajectories for robots $1, \ldots, i$ that are satisfying and $S^{>i}$-avoiding. ■

As we can see from the constructive proof of Theorem 2, the instances that admit $S^{>i}$-avoiding and $G^{<i}$-avoiding paths for each robot can be solved by a simple sequential algorithm that navigates all robots one-after-another along their shortest $S^{>i}$-avoiding and $G^{<i}$-avoiding paths. Albeit simple, such an approach never lets two robots to move concurrently and thus it typically generates solutions of poor quality. The solution quality can be improved if we adopt the prioritized planning approach and find for each robot a best $S^{>i}$-avoiding trajectory that avoids conflicts with higher-priority robots.

Revised Prioritized Planning

We propose a Revised version of Prioritized Planning (RPP) that uses the insights from the preceding discussion and plans the trajectory of each robot so that both a) start position of all lower-priority robots are avoided and b) conflicts with higher-priority robots are avoided. The pseudocode of RPP is listed in Algorithm 2.

Properties: The RPP algorithm inherits the termination and soundness properties from the PP algorithm. The algorithm terminates successfully in n iterations if a trajectory for each

Algorithm 2: Revised Prioritized Planning

```
1 Algorithm RPP
2     Δ ← ∅;
3     for i ← 1 ... n do
4         S ← ⋃_{j>i} S^j
5         π_i ← Best-traj(W \ S, Δ);
6         if π_i = ∅ then
7             report failure and terminate;
8         Δ ← Δ ∪ R_i^Δ(π_i);
```

robot has been found. The algorithm terminates with failure at iteration $i < n$ if there is a robot i for whom a satisfying trajectory in $\mathcal{W} \setminus S$ has not been found.

In general, it is not guaranteed that a trajectory that avoids both start positions of lower-priority robots and regions occupied by higher-priority robots will exist for each robot and thus the algorithm may fail to provide a solution to a solvable problem instance. Consider e.g. the example in Figure 1 once again. However, for the instances characterized by the following condition, the solution is guaranteed exists and RPP will find it.

Corollary 3. *If there is a $S^{>i}$-avoiding, $G^{<i}$-avoiding satisfying path for every robot i and a complete algorithm is used for the single-robot trajectory planning in* `Best-traj` *function, then RPP is guaranteed to terminate with conflict-free solution.*

Proof: Consider the inductive argument from the proof of Theorem 2. The argument states that at every iteration, there exists a $S^{>i}$-avoiding satisfying trajectory for robot i that avoids all higher-priority robots. Since the single-robot planning algorithm is assumed to be complete, it cannot fail in finding such a trajectory. ∎

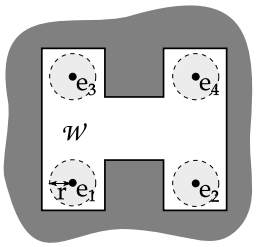
Valid Infrastructures

Consider a situation when one designs a closed multi-robot systems such as a warehouse with a large number of autonomous vehicles. Then, we can exploit the Theorem 2 and Corollary 3 to design the environment and allowed tasks of the robots in such a way that $S^{>i}$-avoiding and $G^{<i}$-avoiding paths will always exists and RPP will be consequently guaranteed to provide a conflict-free solution to every trajectory coordination query. An important class of environments that satisfy the condition of having a $S^{>i}$-avoiding and $G^{<i}$-avoiding paths for every possible task of every robot are *valid infrastructures*.

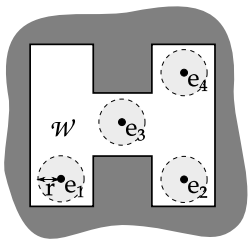
Let $D(x, r)$ be a closed disk centered at x with radius r and $\mathrm{int}_r\ X$ be an r-interior of set X defined as

$$\mathrm{int}_r\ X := \{x : D(x, r) \subseteq X\}.$$

Any path that lies entirely in $\mathrm{int}_r\ X$ will have r-clearance with respect to X, i.e. any point on the path will be at minimum distance r from the closest boundary of X.



(a) Valid infrastructure: The workspace $\mathcal{W}$ and endpoints $\{e_1, e_2, e_3, e_4\}$ for robot having radius r form a valid infrastructure.



(b) Invalid infrastructure: The workspace $\mathcal{W}$ and endpoints $\{e_1, e_2, e_3\}$ do not form a valid infrastructure because there is no path from e_1 to e_2 with $2r$-clearance to e_3 for a robot having radius r.

Figure 4: Valid and invalid infrastructure

We say that a workspace $\mathcal{W}$ together with a set of endpoints E form a *valid infrastructure* for circular robots with maximum radius r if any two endpoints can be connected by a path in workspace $\mathrm{int}_r \left(\mathcal{W} \setminus \bigcup_{e \in E} D(e, r)\right)$, i.e. there must exists a path between any two endpoints with at least r-clearance to the boundary of the workspace and at least $2r$-clearance to any other endpoint. Figure 4 illustrates the concept of a valid infrastructure.

The notion of valid infrastructures follows the structure typically witnessed in man-made environments that are intuitively designed to allow efficient transit of multiple people or vehicles. In such environments, the endpoint locations where people or vehicle stop for long time are separated from the transit area that is reserved for travel between these locations.

In a road network, for example, the endpoints would be the parking places and the system of roads is built in such a way that any two parking places are reachable without crossing any other parking place. Similar structure can be witnessed in offices or factories. The endpoints would be all locations, where people may need to spend longer periods of time, e.g. surroundings of the work desks or machines. As we know from our every day experience, work desks and machines are typically given enough free room around them so that a person working at a desk or a machine does not obstruct people moving between other desks or machines. We can see that real-world environments are indeed often designed as valid infrastructures.

Suppose that a particular workspace $\mathcal{W}$ and a set of endpoints E form a valid infrastructure for robots having radius r. An instance of multi-robot trajectory coordination problem in such an infrastructure would then involve a number of robots traveling from one endpoint to another so that each endpoint is used by at most one robot. From the valid infrastructure property, we know that for each robot there is a path p from its start endpoint to its goal endpoint that avoids all other endpoints in the infrastructure with $2r$ clearance. Since all other robots' start and goal positions lie at some endpoint, the path p is also avoiding start and goal position of every other robot. In other words, for every robot there is a path that is S^{-i}-avoiding and G^{-i}-avoiding, which implies that

the path is also $S^{>i}$-avoiding and $G^{<i}$-avoiding. Therefore, all trajectory coordination queries between the endpoints of a valid infrastructure will be successfully solved by RPP algorithm, given that a complete algorithm is used for the single robot trajectory planning.

Checking Solvability

Now consider an open multi-robot system in which new robot tasks can appear any time and we cannot guarantee that the robots will move only between pre-designed endpoints. Then we can use Theorem 2 to check whether adding a new robot task maintains sequential solvability of the trajectory coordination problem between the existing robots and the new robot. If it does not, then we can decide to reject or delay the adding of new robot task. Corollary 3 gives us sufficient condition that can be used to quickly determine whether a particular problem instance is solvable by RPP without actually running the algorithm. This is done by verifying that $S^{>i}$-avoiding and $G^{<i}$-avoiding satisfying path exists for each robot. Deciding whether such paths exist amounts to planning n spatial paths amidst static obstacles, which is in practice significantly faster than planning n spatio-temporal trajectories amidst dynamic obstacles which would have to be done if RPP is executed.

Limitations

We have shown that there is a class of instances that RPP completely covers, but PP does not. However, outside this class we can find instances that PP solves, but RPP does not. Consider the following scenario:

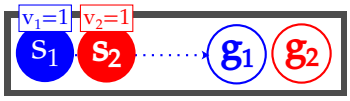


Figure 5: Robot 1 travels from s_1 to g_1, robot 2 travels from s_2 to g_2. Assume that both robots can travel at the same maximum speed. Robot 1 searches for a trajectory that avoids start position of robot 2; such a trajectory does not exist and thus RPP finishes with failure. Note that PP will successfully find a solution to this instance: Robot 1 will plan a trajectory that follows straight line at maximum speed. Robot 2 will search for a trajectory that avoids the trajectory of robot 1 and finds that it suffices to travel at maximum speed to its goal g_2.

None of the algorithm is therefore superior to the other in terms of instance coverage.

Further, since RPP avoid start regions preemptively, even when they can be safely passed through, the solutions generated by RPP tend to be be slightly longer than the ones generated by PP. This can be demonstrated in the following scenario:

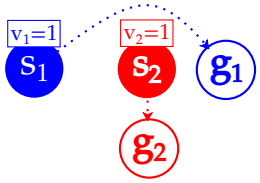


Figure 6: When RPP searches for a trajectory for robot 1 it has to avoid start position of robot 2, resulting in the curved trajectory as depicted in the picture. On the other hand, PP would generate a shorter straight-line trajectory connecting start and destination of robot 1.

We can see that despite the theoretical guarantees of RPP, there exist situations in which PP would be a more appropriate choice than RPP.

IV. Decentralized Algorithms

Imagine a multi-robot system consisting of a large number of heterogeneous autonomous robots. In such a scenario, a decentralized implementation of (revised) prioritized planning may be more desirable than a centralized one. In a decentralized implementation, each robot runs its own instance of the algorithm and exchanges messages with the other robots according to a prescribed communication protocol. If an inconsistency is detected by a robot, then it recomputes the best trajectory for itself using its own on-board computation resources. The process should eventually converge to a state where all robots hold mutually conflict-free trajectories.

An advantage of such an approach is that several robots often end up computing their trajectories in parallel and thus a conflict-free solution is usually computed faster. Another advantage for multi-robot systems with heterogeneous robots is that the kinematic and other potentially implicit constraints on the trajectory of a particular robot stay local to that robot and do not need to be formalized nor communicated, which simplifies the design of the communication protocol and allows each robot to use a custom robot-specific planner for planning its trajectory.

Synchronized Decentralized Implementation

A decentralized implementation of classical prioritized planning scheme, where robots concurrently proceed in synchronize rounds, has been first presented by Velagapudi et al. in [14]. We will use their approach as a baseline decentralized implementation of (revised) prioritized planning and denote the resulting algorithm as *synchronized decentralized implementation of (revised) prioritized planning,* SD-(R)PP.

The algorithm proceeds in synchronized rounds. During every round, each robot ensures that its current trajectory is consistent with the trajectories of higher-priority robots from the previous round. If the current trajectory is consistent, then the robot keeps its current trajectory and remains silent. Otherwise, it finds a new consistent trajectory for itself and broadcasts the trajectory to all other robots. When a robot finishes its computation in the current round, then it waits for all

other robots to finish the round and all robots simultaneously proceed to the next round. The algorithm successfully finishes if none of the robots changes its current trajectory during a single round. The SD-PP algorithm finishes with failure if there is a robot that fails to find a trajectory that avoids the higher-priority robots following their respective trajectories. The SD-RPP algorithm, on the other hand, finishes with failure if there is a robot that fails to find a satisfying trajectory that avoids the start positions of lower-priority robots. The pseudocode of SD-(R)PP is listed in Algorithm 3.

Algorithm 3: Synchronized Decentralized Implementation of (Revised) Prioritized Planning. Pseudocode for robot i

Algorithm `SD-(R)PP`
  $\pi_i \leftarrow \emptyset$;
  $H_i \leftarrow \emptyset$;
  $S \leftarrow \begin{cases} \emptyset & \text{for SD-PP} \\ \bigcup_{j>i} S^j & \text{for SD-RPP} \end{cases}$;
  repeat
    $\pi^* \leftarrow$ `Find-consistent` $(\pi_i, \mathcal{W} \setminus S, \Delta(H_i))$;
    if $\pi^* = \emptyset$ **then**
      report failure and terminate;
    else if $\pi^* \neq \pi_i$ **then**
      $\pi_i \leftarrow \pi^*$;
      broadcast $\mathrm{INFORM}(i, R_i^{\Delta}(\pi^*))$;
    wait for INFORM messages from all other robots, wait for all other robots to finish processing INFORM messages ;
  until *not global termination detected*;
Handle-message $\mathrm{INFORM}(j, \Delta_j)$
  if $j < i$ **then**
    $H_i \leftarrow \left(H_i \setminus \left\{(j, \Delta_j') : (j, \Delta_j') \in H_i\right\}\right) \cup \{(j, \Delta_j)\}$;
Function `Find-consistent` $(\pi, \mathcal{W}, \Delta)$
  if $\pi = \emptyset \vee \neg \mathrm{consistent}_i(\pi, \Delta)$ **then**
    $\pi^* \leftarrow$ `Best-traj`$_i(\mathcal{W}, \Delta)$;
    return π^*;
  else
    return π;

In SD-(R)PP, each robot i maintains a database of space-time regions occupied by higher-priority robots. We call such a database a trajectory store and model it as a set of pairs $H_i = \{(j, \Delta_j)\}$, where Δ_j is the space-time region occupied by robot j.

Function $\Delta(H)$ represents the region of the space-time occupied by all robots stored in a trajectory store H:

$$\Delta(H) := \bigcup_{(j,\Delta_j) \in H} \Delta_j.$$

Further, we use a predicate $\mathrm{consistent_i}(\pi, \Delta)$ to express that robot i following the trajectory π is collision-free against dynamic obstacles Δ, defined as:

$$\mathrm{consistent}_i(\pi, \Delta) \quad := \quad R_i^{\Delta}(\pi) \ \cap \ \Delta \quad = \quad \emptyset.$$

Properties: In order to facilitate and simplify exposition of the later introduced asynchronous algorithm, we developed an alternative proof of termination of the SD-(R)PP algorithm, which deviates from the original one devised by the authors of SD-PP in [14]. Further, in this section we show that SD-(R)PP inherits the soundness and completeness properties from its respective centralized counterpart.

The SD-(R)PP algorithm is guaranteed to terminate. First we need to define what does termination mean for a decentralized algorithm. A decentralized algorithm

- **terminates** when all robots stop computing,
- **terminates with failure** if it terminates and there is at least one robot that reported failure during the computation,
- **terminates successfully** if it terminates and does not terminate with failure.

To show that SD-(R)PP terminates, we first show that robots running SD-(R)PP cannot exchange messages forever.

Proposition 4. *All robots running SD-(R)PP algorithm eventually stop sending INFORM messages.*

Proof: We will proceed by induction on the robot priority i.
Inductive hypothesis: Robots $1, \ldots, i-1$ eventually stop sending messages.
Base step (robot 1):
Robot 1 is the highest-priority robot and as such it does not receive any message from a higher-priority robot. Therefore, its trajectory store will stay empty. During the initialization, the robot 1 either successfully finds its initial trajectory and broadcasts a single message or reports a failure and terminates. Since its trajectory store is empty, its initial trajectory will never become inconsistent, the robot will therefore never replan and send any further INFORM message.
Induction step (robot i):
From the inductive hypothesis we know that each of the robots $1, \ldots, i-1$ eventually stops broadcasting messages. After the last message from the robots $1, \ldots, i-1$ has been received by robot i, its trajectory store gets updated for the last time, since from our assumption there are no more messages from higher-priority robots. After trajectory store changes for the last time, the robot either a) keeps its current trajectory if it is consistent with the last trajectory store, b) finds a new consistent trajectory or c) terminates with failure. In cases a) and b), the current trajectory will never become inconsistent again because trajectory store does not change anymore and thus robot will never have to replan and communicate the new trajectory. In case c), the robot has terminated and thus it won't send any further messages. We see that after robots $1, \ldots, i-1$ stopped sending messages, also robot i eventually stops sending messages. ■

Corollary 5. *SD-(R)PP terminates.*

Proof: We know that all robots in the system will eventually stop sending messages. Assume that the last message is broadcast during round k. In the round $k+1$, no robot

changes its trajectory since otherwise a message would have to be broadcast which is a contradiction. If no robot changes its trajectory during the round, the global termination condition is satisfied and the system terminates. ■

Unless a failure is reported by one of the robots, the solution computed when SD-(R)PP terminates is sound:

Proposition 6. *When SD-(R)PP successfully terminates, then all robots hold trajectories that are mutually conflict-free.*

Proof: Let π_i be the trajectory of robot i after the algorithm terminated. We need to show that

$$\forall i, j : \; i \neq j \Rightarrow \pi_i \text{ and } \pi_j \text{ are conflict-free.}$$

Take two arbitrary, but different robots i,j. Since the conflict-free relation is symmetrical, we can assume $j < i$ w.l.o.g. If robot i stopped computing without failure, then it must have received the INFORM message from higher-priority robot j carrying its last trajectory π_j at some point before its termination. Since there are no further INFORM messages broadcast by robot j, the trajectory store of robot i will contain π_j from that point on. Every trajectory returned by `Find-consistent` function for robot i from that point on will be conflict-free with π_j and thus also its last trajectory π_i will be conflict-free with π_j. ■

The synchronized decentralized implementation of PP and RPP inherit completeness properties from the respective centralized implementations. In general, both SD-PP and SD-RPP are incomplete. However if $S^{>i}$-avoiding and $G^{<i}$-avoiding satisfying path exists for each robot, then the SD-RPP is guaranteed to terminate successfully.

Lemma 7. *If there is a $S^{>i}$-avoiding, $G^{<i}$-avoiding satisfying path for every robot i and a complete algorithm is used for the single-robot trajectory planning in* `Best-traj` *function, then SD-RPP is guaranteed to terminate with a conflict-free solution.*

Proof: The argument used in the proof of Theorem 3, which shows that RPP will never fail during planning, can be extended to decentralized implementations of RPP: Take arbitrary replanning request for robot i. All trajectories of each higher-priority robot $j < i$ have been generated to be $S^{>j}$-avoiding and thus such trajectory will be also S^i-avoiding. All trajectories in the trajectory store of robot i are therefore S^i-avoiding. An $S^{>i}$-avoiding satisfying trajectory consistent with trajectories of higher-priority robots can be constructed as follows. Wait at start position s_i until all higher-priority robots reach their goal position and then follow the $S^{>i}$-avoiding $G^{<i}$-avoiding satisfying path from the assumption. Since such a trajectory is guaranteed to exist for robot i and a complete replanning algorithm is used, the replanning cannot report failure. The algorithm must terminate with success. ■

Asynchronous Decentralized Implementation

Due to its synchronous nature, the SD-(R)PP algorithm does not fully exploit the computational resources distributed among individual robots. In every iteration, the robots that finished their trajectory planning routine sooner, or did not have to re-plan at all, idle while waiting for the slower computing robots in that round, even though they could use the time to resolve some of the conflicts they have among themselves and speed up the overall process. An example of a situation, where the asynchronous algorithm would be beneficial is illustrated in Figure 7.

To deal with such an inefficiency, we propose an asynchronous decentralized implementation of the (revised) prioritized planning scheme, abbreviated as AD-(R)PP. The pseudocode code of AD-(R)PP is exposed in Algorithm 4. The asynchronous algorithm replaces the concept of globally synchronized rounds (while loop in Algorithm 3) by a reactive approach in which every robot reacts merely to incoming INFORM messages. Upon receiving an INFORM message (**Handle-message** $\text{INFORM}(j, \Delta_j)$ routine in Algorithm 4), the robot simply replaces the information about the trajectory of the sender robot in its trajectory store and checks whether its current trajectory is still consistent with the new contents of its trajectory store. If the current trajectory is inconsistent, the robot triggers replanning and inform other robots about its new trajectory, otherwise the robot keeps its current trajectory and remains silent.

Algorithm 4: Asynchronous Decentralized Implementation of (Revised) Prioritized Planning

Algorithm `AD-(R)PP`
  $\pi_i \leftarrow \emptyset$;
  $H_i \leftarrow \emptyset$;
  $S \leftarrow \begin{cases} \emptyset & \text{for AD-PP} \\ \bigcup_{j>i} S^j & \text{for AD-RPP} \end{cases}$;
  $\pi_i \leftarrow$ `Find-consistent` $(\pi_i, \mathcal{W} \backslash S, \Delta(H_i))$;
  if $\pi_i = \emptyset$ **then**
    report failure and terminate;
  else
    broadcast $\text{INFORM}(i, R_i^{\Delta}(\pi_i))$;
    wait for global termination;
Handle-message $\text{INFORM}(j, \Delta_j)$
  if $j < i$ **then**
    $H_i \leftarrow \left(H_i \setminus \left\{(j, \Delta_j') : (j, \Delta_j') \in H_i\right\}\right) \cup \{(j, \Delta_j)\}$;
    $\pi^* \leftarrow$ `Find-consistent` $(\pi_i, \mathcal{W} \setminus S, \Delta(H_i))$;
    if $\pi^* = \emptyset$ **then**
      report failure and terminate;
    else if $\pi^* \neq \pi_i$ **then**
      $\pi_i \leftarrow \pi^*$;
      broadcast $\text{INFORM}(i, R_i^{\Delta}(\pi^*))$;

Properties: AD-(R)PP inherits all the desirable properties from its synchronized and centralized coutnerparts, i.e. it terminates and if it terminates with success than all the robots will hold conflict-free trajectories. Further, AD-RPP is guaranteed to solve instances that admit $S^{>i}$-avoiding and $G^{<i}$-avoiding path for each robot.

Proposition 8. *AD-(R)PP terminates.*

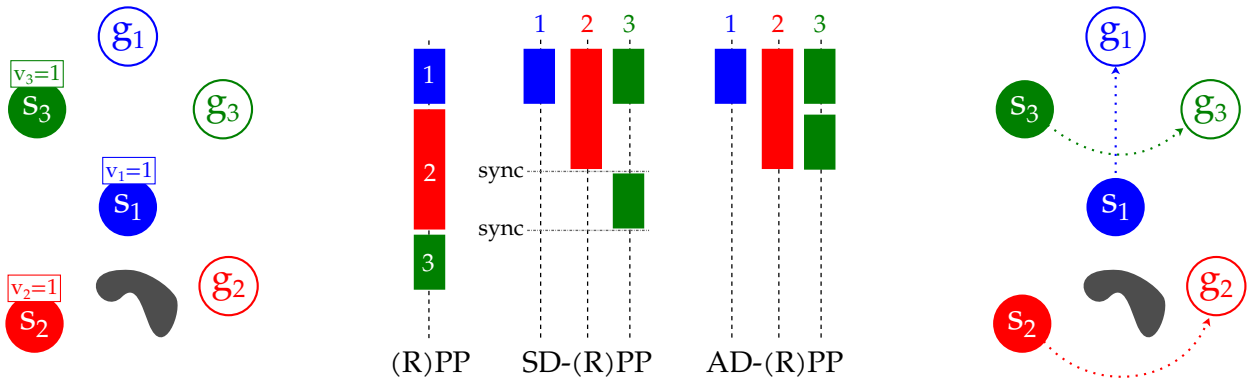


Figure 7: Example problem in which AD-(R)PP converges faster than SD-(R)PP. **Left:** The task of robots 1, 2, and 3. **Middle:** Sequence diagrams showing the planning process in (R)PP, SD-(R)PP and AD-(R)PP. Suppose that robot 2 needs to plan longer, because it has to plan around the gray obstacle. In SD-(R)PP, the robot 3 starts resolving the conflict with robot 1 only when robot 2 has finished computing its trajectory in the first round. In AD-(R)PP, robot 3 stars resolving the conflict with robot 1 immediately after it becomes aware of it, therefore it finds the solution faster. **Right:** The final solution.

Proof: Recall that the inductive argument demonstrating that robots running SD-(R)PP will eventually stop sending messages (Lemma 4) does not make use of the synchronization points in SD-(R)PP and thus it is also valid for AD-(R)PP. By this argument, we know that there is a finite number of messages being sent. We can observe that a robot running AD-(R)PP performs computation only during initialization or when it processes an incoming message. When all robots process their last incoming messages, the system terminates. ■

Proposition 9. *When AD-(R)PP successfully terminates, then all robots hold trajectories that are mutually conflict-free.*

Proof: The proof of soundness of SD-(R)PP (Proposition 6) is directly applicable also to AD-(R)PP. ■

Proposition 10. *If $S^{>i}$-avoiding, $G^{<i}$-avoiding satisfying path exists for every robot i and a complete algorithm is used for the single-robot trajectory planning in* `Best-traj` *function, then AD-RPP terminates.*

Proof: The proof of Proposition 7, where this property is demonstrated to hold for SD-RPP, is directly applicable also for AD-RPP. ■

V. Experiments

We compare the performance of PP, RPP, SD-PP, SD-RPP, AD-PP, AD-RPP in terms of coverage, runtime, communication complexity and solution quality. The comparison was performed in three real-world environments (see Figure 8a, 9a and 10a). For each of the environments we generated two sets of problem instances: 1) In *free-formed tasks* instance set, each robot is assigned a task to move from a randomly selected start position to a randomly selected goal position. 2) In *infrastructure tasks* instance set, we generated a set of endpoints that together with a particular roadmap discretization of the environment form a valid infrastructure; each robot is then assigned a random endpoint as a start position and a randomly chosen endpoint as a destination position.

For the decentralized algorithms (SD-PP, SD-RPP, AD-PP and AD-RPP), we assume that each robot uses its own on-board CPU to compute its trajectory. To measure the runtime characteristics of the execution of decentralized algorithms, we emulate the concurrent execution of the algorithms using a discrete-event simulation. The simulation measures the execution time of each message handling and uses the information to simulate the concurrent execution of the decentralized algorithm as if it is executed on n independent CPUs, where n is the number of robots. The concurrent process execution simulator was implemented using Alite multi-agent simulation toolkit. The source code of the entire experimental setup can be downloaded at http://agents.cz/~cap/adpp/.

All compared algorithms use identical best trajectory planner. The best trajectory for each robots is obtained by searching a roadmap extended with a discretized time-dimension using A* algorithm, where the heuristic is the shortest path on the graph from the given node to the goal node when the dynamic obstacles are ignored.

The experiments have been performed on AMD Opteron 8356 2.3GHz, 8 GB RAM. For each algorithm we measure the following characteristics:

Coverage: We waited before each algorithm returns either success or failure and counted the number of instances each of the algorithm successfully solved.

The following characteristics were measured only on instances that were solved by all compared algorithms:

Time to solution: We measured the wall-clock runtime needed to compute a solution. For the centralized planner we recorded the time of termination of the centralized planner. For the decentralized algorithms we recorded the time when the last robot detected global termination of the computation.

Speed-up: In order to be able to easier judge the effect of asynchronous execution in AD-(R)PP algorithm we also compute the speed-up ratio for both decentralized algorithms over their centralized counterparts. The speed-up for algorithm

A on instance i is computed as

$$\frac{\text{runtime of centralized variant of alg. } A\text{on instance } i}{\text{runtime of alg. } A \text{ on instance } i},$$

where the centralized variant of AD-PP and SD-PP is PP, and the centralized variant of AD-RPP and SD-RPP is RPP.

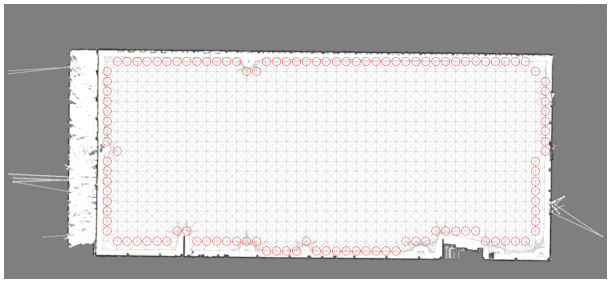
Replannings/Communication: Every time a robot running a decentralized algorithm adopts a new trajectory (replans), the trajectory is broadcast to all other robots. Therefore the number of replannings directly corresponds to the number of INFORM message broadcast in the system.

Prolongation: The objective criterion we minimize is the sum of goal arrival times for each robot. We measure the duration of the trajectory for each robot and compute the prolongation coefficient for each instance as

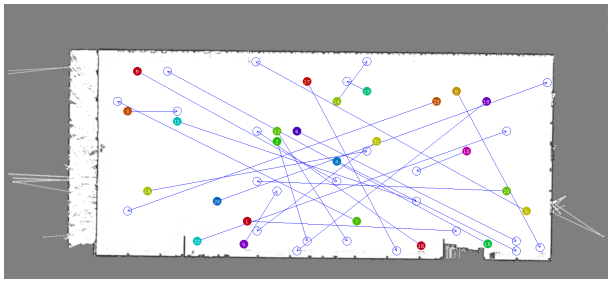
$$\text{prolongation of alg. } A \text{ on instance } i = \frac{\sum_{i=1}^{n} t_i^A - t_i'}{\sum_{i=1}^{n} t_i'}, \quad (1)$$

where t_i^A is the time robot i needs to reach its goal position when it follows the trajectory computed by the algorithm A and t_i' is the time the robot i would have needed to reach its goal following the shortest path on the roadmap if the collisions with other robots were ignored.

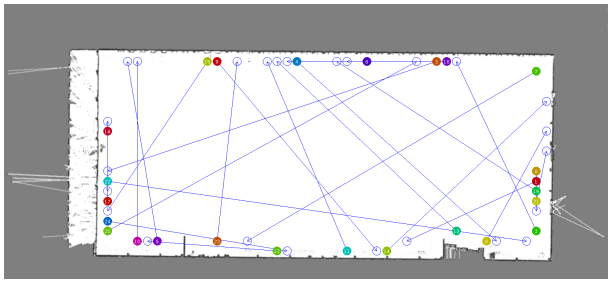
Comparison with reactive planning: In order to evaluate the practical advantages of the proposed planning approaches, we also compare their coverage with the popular reactive technique ORCA in our environments. ORCA is a control approach that continuously observes positions and velocities of other robots in a defined neighborhood. Should any potential collision be detected, a linear program is used to compute a new collision averting velocity that the robot should follow. If there are no eminent collisions, the robot follows its preferred velocity. In our implementation the preferred velocity points at the globally shortest path from the robots current position to the goal.



(a) *Empty-hall* Environment. The roadmap used for planning is depicted in gray. Infrastructure endpoints are shown in red.



(b) Example free-formed tasks for 25 robots. Task of each robot shown in blue.



(c) Example infrastructure tasks for 25 robots. Task of each robot shown in blue.

Figure 8: Empty hall environment

Environments

Empty-hall environment: The *empty-hall* environment and the roadmap used for trajectory planning in the environment are depicted in Figure 8a. For both instance sets and each number of robots ranging from $n = 1$ to $n = 50$, we generated 25 random instances of the problem containing the given number of robots.

Office Corridor environment: The *office corridor* environment is based on the laser rangefinder log of Cartesium building at the University of Bremen. We thank Cyrill Stachniss for providing the data through the Robotics Data Set Repository [5]. The environment and the roadmap used for trajectory planning in the environment are depicted in Figure 9a. For both instance sets and each number of robots ranging from $n = 1$ to $n = 30$, we generated 25 random problem instances containing the given number of robots.

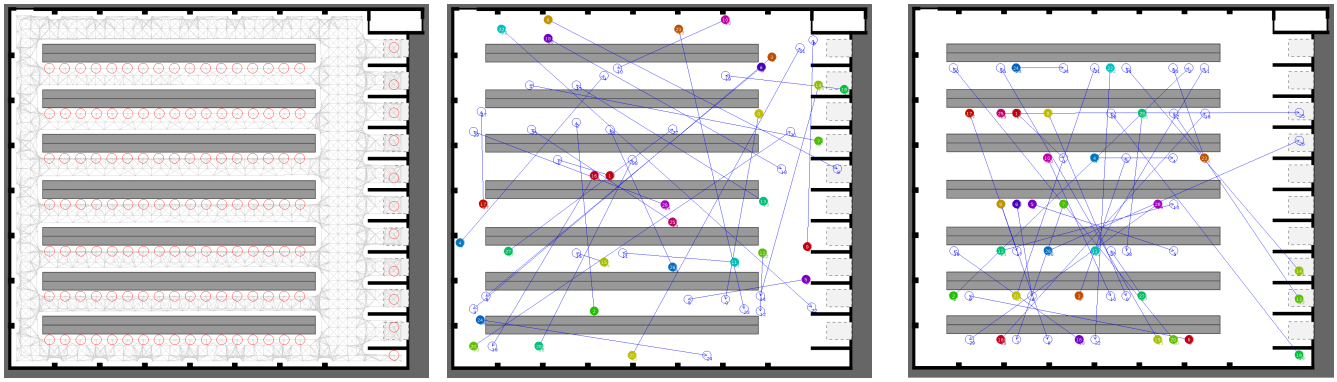
Warehouse environment: The map of the *warehouse* environment and the roadmap used for trajectory planning are depicted in Figure 10a. For both instance sets and each number of robots ranging from $n = 1$ to $n = 60$, we generated 25 random problem instances containing the given number of robots. The infrastructure tasks instance set represents a scenario of an automated logistic center where robots move goods between the gates and the storage shelves.

Results

The results of the comparison in the three test environments for free-formed and infrastructure tasks are plotted in Figure 11 and Figure 12 respectively.

Note that in the corridor environment and warehouse environment with free-formed tasks, all tested algorithms exhibit low success-rate on instances with higher number of robots. Since the plots from these two instance sets are based only on the few instances that all the tested algorithms were able to solve, they are shown only for the sake of completeness and will not be used to draw statistically significant conclusions.

Coverage: For free-formed tasks, all tested algorithms exhibit incomplete coverage of the instance space. Generally, RPP-based algorithms solved fewer instances than the PP-based algorithms. This phenomena can be explained as follows: The failure of PP-based algorithms is due to Type A (robot cannot avoid being run-over by a higher-priority robot)



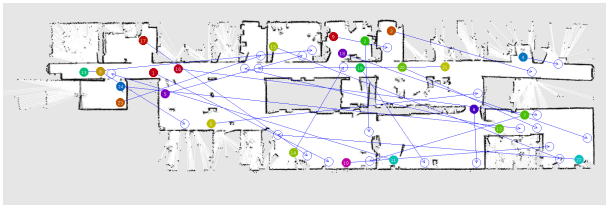
(a) *Warehouse* Environment. The roadmap used for planning is depicted in gray. Infrastructure endpoints are shown in red.

(b) Example free-formed tasks for 30 robots. Task of each robot shown in blue.

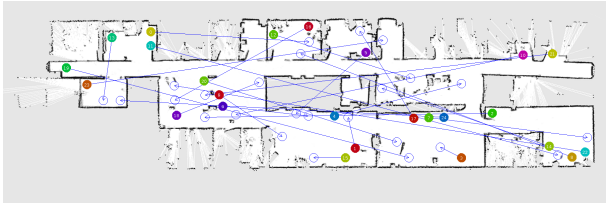
(c) Example infrastructure tasks for 30 robots. Task of each robot shown in blue.

Figure 10: Warehouse environment

(a) *Office Corridor* Environment. The roadmap used for planning is depicted in gray. Infrastructure endpoints are shown in red.



(b) Example free-formed tasks for 25 robots. Task of each robot shown in blue.



(c) Example infrastructure tasks for 25 robots. Task of each robot shown in blue.

Figure 9: Office Corridor environment

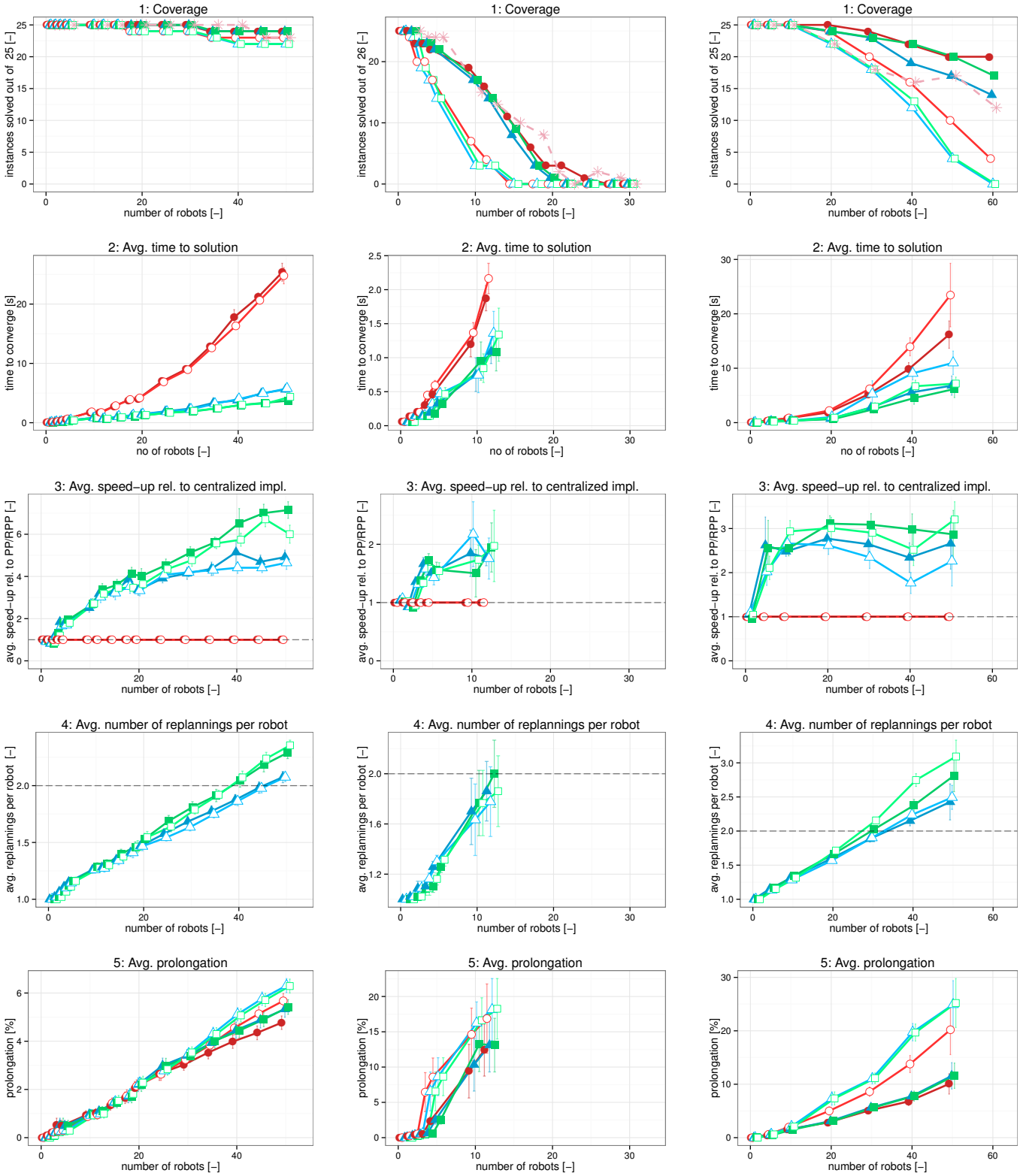
or Type B (all paths to the goal of a robot are blocked by higher-priority robots at their goal position) conflicts. The failure of a RPP-based algorithms can be caused either by non-existence of path that avoids start-positions of higher-priority robots or by Type-B conflict. It is more likely, however, that a start-avoiding path will not exists (since it is a static condition that holds at all times) than that a Type B conflict will occur, thus the lower success rate of RPP-based algorithms.

For infrastructure tasks, the RPP-based instances show full-instance coverage in accordance with our theoretical analysis. Further, we can see (best in Figure 12a-1) that some of those instances remain unsolved both by PP-based algorithms and ORCA.

Time to solution/speed-up: The asynchronous decentralized implementation of both PP and RPP consistently achieves higher speed-up than the synchronized implementation in accordance with our prediction. The higher speed-up is exhibited on instances with higher number of robots, where it is more likely that several independent conflict clusters will occur. On such instances, it is often beneficial that the conflict clusters can develop at different pace and thus converge faster. The phenomena can be seen clearly in Figures 11a-3, 12a-3, 12b-3, and 12a-3.

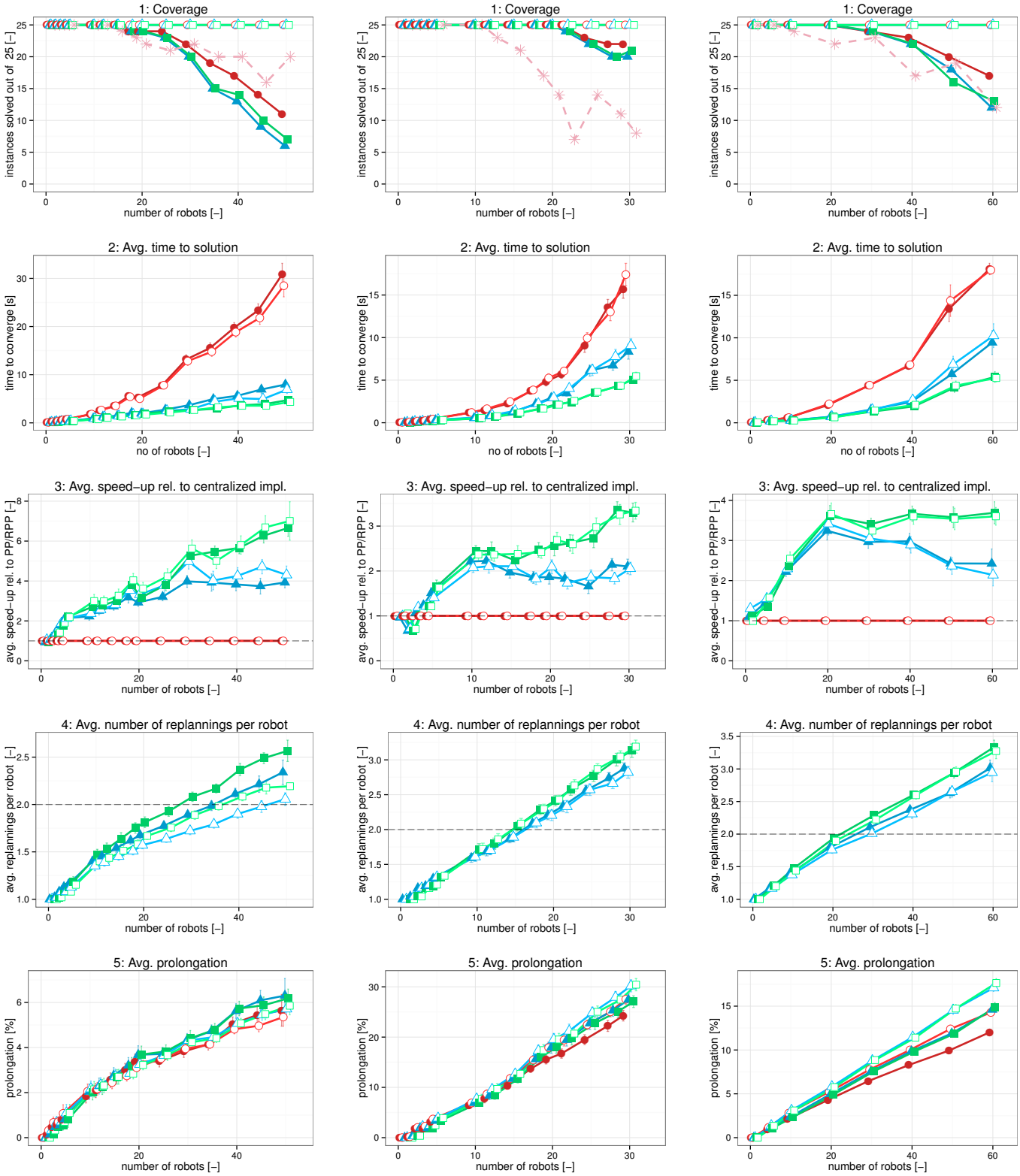
Replannings/Communication: AD-(R)PP broadcast higher number of messages than SD-(R)PP. To see how this can be explained, suppose that at some point of the computation new conflicts arise between the trajectory of one particular robot and the trajectories of two other higher-priority robots. If the two conflicts occur in a single round, SD-(R)PP solves both conflicts during one replanning at the end of the round and therefore broadcasts only a single INFORM message. However, in such a situation AD-(R)PP may need to replan twice because it triggers replanning immediately after each of the conflicts is detected and thus it will broadcast two INFORM messages.

Prolongation: There are two phenomena influencing the quality of returned solutions. First, RPP-based algorithms generate slightly longer trajectories than PP-based algorithms. This is due to the fact that RPP preemptively avoids start positions of the lower-priority robots. Second, decentralized approaches generate slightly longer trajectories than the centralized approaches. The reason is the replanning condition used by the decentralized algorithms. The condition states that a robot should replan its trajectory only if the trajectory is inconsistent with the trajectories of other robots. Thus, the robot may receive an updated trajectory from a higher-priority robot that allows an improvement in its current trajectory, but since its current trajectory may be still consistent, the robot will not exploit such an opportunity for an improvement.



(a) **Empty hall environment** (b) **Office corridor environment** (c) **Warehouse environment**

Figure 11: Results: Free-formed tasks (Bars indicate standard error)



(a) **Empty hall environment** (b) **Office corridor environment** (c) **Warehouse environment**

Figure 12: Results: Infrastructure tasks (Bars indicate standard error)

VI. Conclusion

Prioritized planning is a practical approach for multi-robot trajectory planning. In this paper, we have summarized properties and compared performance of six different algorithms employing the idea of prioritized planning. While PP and SD-PP are existing algorithms that have previously appeared in the literature, the remaining four algorithms are our novel contributions.

In particular, we have proposed a revised version of prioritize planning (RPP) and proved that this algorithm is guaranteed to provide solution if there is a path for every robot that reaches its goal position, avoids start positions of lower-priority robots and avoids goal positions of higher-priority robots. We have shown that this condition is satisfied if the individual robots move between two endpoints of a valid infrastructure. The significance of this result lies in the fact that human-made environments are usually build as valid infrastructures and thus the RPP algorithm can be used to efficiently find coordinated trajectories for the robots operating in such environments. We have experimentally demonstrated that in valid infrastructures, RPP algorithm solves instances that would be otherwise unsolvable by state-of-the-art techniques such as the classical prioritized planning or ORCA.

A decentralized algorithm for the multi-robot trajectory coordination problem is often more desirable because the trajectory planning may be performed locally by each robot and several robots may plan in parallel. We proposed a novel asynchronous decentralized implementation of (revised) prioritized planning scheme and proved that the algorithm is guaranteed to terminate. Further, the asynchronous decentralized implementation of revised prioritized planning is guaranteed to provide a solution under the same conditions as its centralized counterpart and thus it can be used to reliably plan coordinated trajectories in valid infrastructures. Further we have shown that in our test environments the asynchronous approach converges faster than the previously known synchronized approach.

In future, we plan to study extensions of the presented decentralized algorithms to open multi-robot systems with local communication.

Acknowledgements

This research was supported by the Czech Science Foundation (grant No. 13-22125S) and by the Grant Agency of the Czech Technical University in Prague grant SGS14/143/OHK3/2T/13. Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the program "Projects of Large Infrastructure for Research, Development, and Innovations" (LM2010005), is greatly appreciated.

References

[1] M. Bennewitz, W. Burgard, and S. Thrun. Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots. *Robotics and Autonomous Systems*, 41(2):89–99, 2002.

[2] Michal Cap, Peter Novak, Jiri Vokrinek, and Michal Pechoucek. Asynchronous decentralized algorithm for space-time cooperative pathfinding. In *Spatio-Temporal Dynamics Workshop (STeDy), SFB/TR 8 Spatial Cognition Center Report, No. 030-08/2012*, 2012.

[3] Michael Erdmann and Tomas Lozano-Pérez. On multiple moving objects. *Algorithmica*, 2:1419–1424, 1987.

[4] J.E. Hopcroft, J.T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects; pspace- hardness of the "warehouseman's problem". *The International Journal of Robotics Research*, 3(4):76–88, December 1984.

[5] Andrew Howard and Nicholas Roy. The robotics data set repository (radish), 2003.

[6] Venkatraman Narayanan, Mike Phillips, and Maxim Likhachev. Anytime safe interval path planning for dynamic environments. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4708–4715. IEEE, 2012.

[7] David Silver. Cooperative pathfinding. In *AIIDE*, pages 117–122, 2005.

[8] Paul G. Spirakis and Chee-Keng Yap. Strong np-hardness of moving many discs. *Inf. Process. Lett.*, 19(1):55–59, 1984.

[9] Trevor Scott Standley. Finding optimal solutions to cooperative pathfinding problems. In Maria Fox and David Poole, editors, *AAAI*. AAAI Press, 2010.

[10] Trevor Scott Standley and Richard E. Korf. Complete algorithms for cooperative pathfinding problems. In Toby Walsh, editor, *IJCAI*, pages 668–673. IJCAI/AAAI, 2011.

[11] Jur van den Berg and Mark Overmars. Prioritized motion planning for multiple robots. In *IROS*, pages 430–435, 2005.

[12] Jur van den Berg and Mark Overmars. Kinodynamic motion planning on roadmaps in dynamic environments. In *IROS*, pages 4253–4258. IEEE, 2007.

[13] Michal Čáp, Peter Novák, Martin Selecký, Jan Faigl, and Jiří Vokřínek. Asynchronous decentralized prioritized planning for coordination in multi-robot system. In *Intelligent Robots and Systems (IROS), 2013*, 2013.

[14] Prasanna Velagapudi, Katia P. Sycara, and Paul Scerri. Decentralized prioritized planning in large multirobot teams. In *IROS*, pages 4603–4609. IEEE, 2010.

[15] Glenn Wagner and Howie Choset. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219:1 – 24, 2015.