Deliberation Dialogues for Cooperative Pathfinding

Xeryus Stokkel
Primary supervisor: prof. dr. Bart Verheij
Secondary supervisor: prof. dr. Rineke Verbrugge

March 2018

Abstract

Cooperative pathfinding research studies coordination algorithms addressing congestions, deadlocks, and collisions in multi-agent systems. In typical algorithms individual agents have no say in resolving conflicts. We propose algorithms in which agents engage in an argumentative dialogue in case of local conflicts, leading to the transparent and fast construction of global solutions. We combine ideas from computational argumentation, multi-agent coordination and continual planning. From computational argumentation we use argumentative deliberation dialogues in which agents discuss and resolve conflicting local plans. From the study of multi-agent coordination we use partial global planning, a distributed method to incrementally create a global plan. Using ideas from continual planning we obtain an online algorithm in which planning and execution are interleaved. We show that our algorithms generally solve cooperative pathfinding problems faster than a state of the art complete and optimal algorithm, at the cost of slightly longer path lengths and gaining the explanatory power of argumentation dialogues. An online version of our algorithm is the fastest with the trade-off that it has the lowest quality paths.

Contents

1 Introduction

When multiple agents have to find their way through a shared space they have to find paths around obstacles while they also need to ensure that they do not collide with each other. This problem is considerably more complex than finding a path for a single agent [14]. Even when agents can prevent collisions then it is still possible that congestions or even deadlocks may occur. Agents need to be able to get to their destination as soon as possible so congestions and deadlocks are undesirable. To avoid them there is a need for coordination between the agents. Cooperative cooperative pathfinding finds its application in robotics, aviation, road traffic management, crowd simulations, and video games [30].

The most straightforward approach to the cooperative pathfinding problem is to search the Cartesian product of the state spaces of all agents. This approach is computationally inefficient [14, 27] as the time to find a solution grows exponential in the number of agents. A common approach to speed up the search is to impose a hierarchy on the agents by assigning them a unique priority. Agents plan a path to their destination in descending order of priority. When it is an agent's turn to plan their path they have to consider agents with a higher priority as a moving obstacle. This means that they have to avoid planning any movements that conflict with those of higher priority agents. Both of these approaches result in abstract solutions; there is often no clear reason why a particular solution was the one arrived at. The algorithm has found a set of conflict free paths that work as a solution but it doesn't give any indication about the considerations of why it is a good plan.

These two common approaches to solving the cooperative pathfinding problem both rely on a central processor [7]. The first is a category of centralized methods that use a central processor to create a plan for all the agents. The other category requires that a central processor determines a priority ordering that the agents have to adhere to. After this has been done then the calculation of the plans for the agents can be decoupled. This allows the agents to make their individual plan on their own processor. During this decoupled planning they need to communicate with each other about their paths but they do not require a central point of communication to do so. Next to the centralized and decoupled methods there are also fully decentralized approaches. With these there is no central processor that can be a single point of failure. As a trade-off these methods usually have no global view of the problem. This means that agents can make decisions early on that will lead to congestions or deadlocks at a later point in time without any agent noticing at the time that the decision was made.

Methods of decentralized coordination have been developed by the field of computational argumentation. Formal models of argumentation have been used in Artificial Intelligence in expert systems, multi-agent systems and law [32, 26]. An important concept in computational argumentation is that of defeasible reasoning [9]: the conclusion that can be drawn from a set of premises does not need to hold when additional premises are added. This is in contrast with classical logic where adding additional premises will never invalidate a conclusion. Defeasible reasoning allow arguments to be made for or against a conclusion. Arguments can also support or attack each other and thereby strengthen or weaken a case for a conclusion.

Commonly computational argumentation in a multi-agent systems is modelled as a dialogue game. In such a dialogue game the agents represent the players and the game rules prescribes how the dialogue should occur [33]. There are rules about what arguments agents can put forward, when they are allowed to do so, and there can be rules about which agent gets to speak when. Most forms of dialogue games also have rules about when the dialogue is finished and which agent(s) have won if applicable. These dialogues can be used to give reasons about why a group of agents decided to take a certain course of action. So they can be used to remove some of the abstractness of cooperative pathfinding algorithms by showing why a solution is preferred over other solutions. Conventional algorithms deliver a solution without indicating why that solution is preferred over others.

Global cooperation between agents without a central processor is a difficult task. There are methods that do achieve a global plan without any single agent being vital to create it. Partial global planning has been used in distributed sensor networks to distribute and coordinate tasks among the nodes that make up the network [11]. The nodes create their individual plans without regard for each other. They will then exchange information on their plans and adapt them to better coordinate their activities. Nodes can even take over each others tasks to spread the computational load. Coordination is not rigid and nodes have some freedom in how they execute their plan if circumstances change without having to re-coordinate with the other nodes. None of the involved nodes ever has a global view but the end result is a plan that is globally coordinated with each node holding a part of the global plan.

This method of constructing a global plan from local views can also be applied to cooperative pathfinding. Agents only have to coordinate with those agents that they have a conflicting path with. The freedom in planning allows agents to find an alternative path without having to update all other agents. Other agents that have previously been coordinated with don't need to update their plan as a response. This is only necessary when new conflicts arise because of the alternative path. This allows for a truly decentralised approach where agents only communicate with other agents when they have to. There is also no need to wait for a central processor to tell agents what to do. At the same time plans are well coordinated and there is a global view, something that other decentralized cooperative pathfinding algorithms lack.

Dialogues can be used in cooperative pathfinding by applying techniques from partial global planning. When agents have a conflict then they need to cooperate to avoid conflicts. They can do so by starting a dialogue in which they share and evaluate different hypothesis to solve the conflict. A hypothesis consists of a priority ordering for the agents that are involved in a conflict. The hypotheses offered will be discussed and evaluated in the dialogue and the agents will give their preference for each proposal. The proposal which is most preferred is used as the solution to the conflict. All agents involved in the dialogue adapt the hypothesis. Next they update their plans so that there are no conflicts between them any more. This means that there are many small local changes to an agent's position in the hierarchy and therefore also in their plans. The end result is a global solution to the cooperative pathfinding problem without any agent having known it explicitly. There is also no single agent which has been vital to its calculation like in a centralized approach.

By combining deliberation dialogues and partial global planning we can develop an algorithm that is able to overcome the weaknesses which other coop-

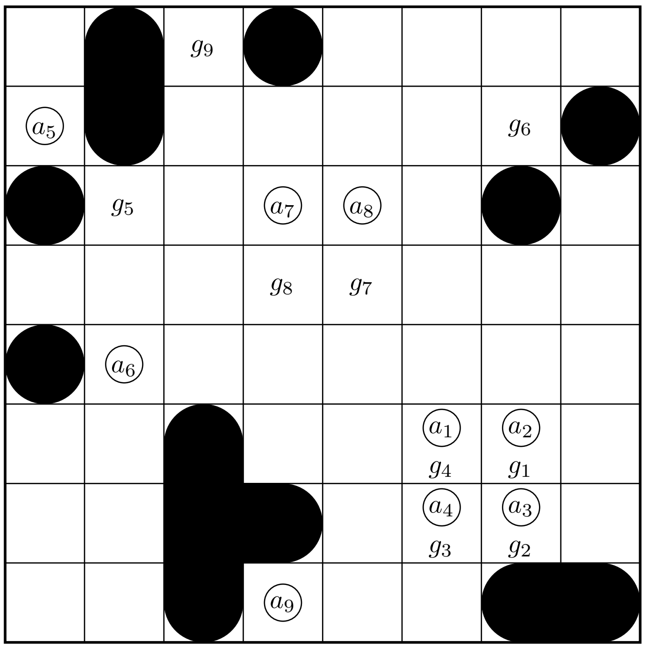
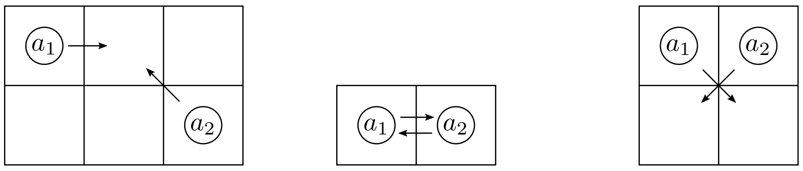


Figure 1: A small space shared by some agents. Obstacles are black, agents are circles inscribed with the agent's number (a_i). The destination for agent a_i is given by g_i.

erative pathfinding algorithms have. Using the decoupled method as a starting point we employ partial global planning to remove the reliance on a central processor to determine the hierarchy. This also prevents a common pitfall of decentralized methods which are not able to coordinate plans on a global level. So we essentially achieve a decentralized algorithm that is able to create a global plan. To enable the use of partial global planning we use deliberation dialogues so that agents can determine a hierarchy in a decentralized fashion. Another benefit of using deliberation dialogues is that it is possible to get reasons why agents settled on a particular hierarchy. This makes it possible to explain to an end-user why the solution is the best solution. This is not possible with conventional cooperative pathfinding algorithms because they only compute a solution according to an algorithm without having any explanatory power.

The rest of this thesis is structured as follows. First, a formal description of the cooperative pathfinding problem is given in Section 2. Previous work in cooperative pathfinding, argumentation and partial global planning is discussed in Section 3. A new method to find conflict-free paths is proposed in Section 4. The method is evaluated and compared to other algorithms in Section 5. Final remarks on the proposed method and its implications are discussed in Section 6.



(a) Moving to the same position. (b) Moving along the same edge. (c) Moving on crossing edges.

Figure 2: Examples of conflicting actions. Agents are circles inscribed with a_i. Their movements are indicated by the arrows starting in the cell they occupy, the action ends in the cell that the arrow points to.

2 Problem formulation

The problem of cooperative pathfinding can be defined as follows. A shared space is divided into discrete cells such that it forms an 8-connected grid. Some of the cells in this grid are static obstacles while the other cells are open. A set of k agents $\{a_1, \ldots, a_k\}$ occupy cells within the grid, the agents have respective goal positions $g_1, \ldots, g_k$. A set of paths need to be found, one for each agent, such that each agent gets to its goal position without colliding with any of the other agents. A path consists of a series of actions. An action can either be to move to one of the eight neighbouring cells or *wait* at the current location. Each time step an agent must do exactly one of these actions. All actions take exactly one time step to execute. An example initial configuration is shown in Figure 1.

The goal is to find one path for each agent so that it reaches its destination in as few actions as possible. The agents can not enter cells with static obstacles nor should agents collide with each other. Each action in a path has unit cost, with the exception of waiting in the goal position. The cost function is then

$$\textsc{cost}(\text{P,Q}) = \begin{cases} 0 & \text{if } P = Q = G \\ 1 & \text{otherwise} \end{cases}$$

where P is the node where the agent is located, Q is the node where the agent moves to, and G is the agent's goal node. A single agent's path has a cost that is the sum of the costs of all its actions. The cost of a solution is defined as the sum of the costs of the paths of the agents. The most appropriate solution to the problem is a solution with minimal cost.

The paths of two agents are not in conflict if and only if at no time step the agents occupy the same cell, agents move along the same edge (swap positions), or agents move along crossing edges. Obstacle cells can be considered as stationary agents. Examples of each of these conflicts are given in Figure 2, it shows that conflicts involving agents moving along the same edge or moving along crossing edges can only occur when agents are in neighbouring cells. A conflict in which agents move to the same cell can happen whenever the agents are at most two actions away from each other. A single action can result in an agent having multiple conflicts at the same time. If Figure 2a had an agent a_3 in the top right cell moving to the bottom middle cell then a_2 would have a conflict with both a_1 and a_3 at the same time but a_1 would only have a conflict with a_2. Agents are allowed to move along a diagonal even when the two cells on the opposing diagonal are blocked, i.e. a_5 in Figure 1 can move to its destination in a single time step. An agent a_i can move to a cell occupied by agent a_j given that a_j will move to a different cell at the same time. Agents a_1, a_2, a_3 and a_4 in Figure 1 can reach their respective destinations by "rotating" clockwise. They can do this in a single time step without requiring any additional empty cells. Agents a_7 and a_8 cannot move to their destinations in a single time step because that would mean that they move along crossing edges at the same time. They can also not swap places because then they would be travelling along the same edge.

3 Related Work

The following sections discuss previous work into cooperative pathfinding, argumentation and coordination. The cooperative pathfinding problem requires that agents are able to coordinate their movement. Several different approaches that achieve this will be discussed below. Computational argumentation has been used in various domains. One of these domains is the construction of a plan for agents, this application of argumentation is known as *practical reasoning*. It can also be used to make plans in a multi-agent system which allows the agents in such a system to coordinate. Argumentation has not yet been used to find solutions for cooperative pathfinding but research in argumentation has been generic enough that it can be applied to a specific application such as cooperative pathfinding. Work in coordination is also discussed to help bridge the gap between argumentation and cooperative pathfinding. We also discuss work in coordination that can be used to achieve greater speed performance.

3.1 Cooperative pathfinding

In the grid world of Figure 1 each agent can take one of $b+1$ actions, where b is the current number of neighbouring cells without static obstacles. There is also a *wait* action where an agent does not move. All cells that are adjacent to the agents current cell are considered to be neighbouring. This includes cells that can be reached by moving diagonally. The naive approach to finding conflict free paths takes the Cartesian product the state spaces of all k agents and searches the new combined state space with a search algorithm like A*. This is also known as the Standard Algorithm [29]. This results in a branching factor of $(b+1)^k$, the branching factor grows exponentially in the number of agents and the problem quickly becomes intractable even with efficient search algorithms like A* [27].

There are a few common strategies that are used to tackle this problem. Centralised methods use one single processor to calculate the paths for all agents. They are often complete: a solution to the problem will be found if one exists. This also means that they are slow. An alternative strategy is to decouple the agents from each other. Each agent plans its own path and a hierarchy is enforced on the agents. Agents with a lower priority need to give way to agents with a higher priority. Decoupled methods sacrifice completeness for speed. They often calculate the priorities at a central processor but can exploit the

Table 1: Comparison of several cooperative pathfinding algorithms.

Method	Category	Complete	Priority	Comm.	Online
OD+ID [29, 30]	Centralized	Yes	No	All	No
ICTS [27]	Centralized	Yes	No	All	No
ADPP [5]	Decoupled	No	Yes	All	No
WHCA* [28]	Decoupled	No	Yes	Window	Yes
DMRCP [35]	Decentralized	No	No	2 nodes	Yes
DiMPP [7]	Decentralized	Yes	Yes	Ring	No
ORCA [31]	Decentralized	No	No	None	Yes

inherent parallelism in multi-agent systems to calculate the paths. There are also decentralized methods that will only solve conflicts when they occur during plan execution. These decentralized methods are often reactive in nature and are not always able to plan far enough into the future to avoid deadlocks and congestions.

An overview of several cooperative pathfinding algorithms is shown in Table 1. It summarises the properties of the algorithms. Each algorithm is discussed in more detail below. Some other aspects than the category, completeness and the assignment of priorities are also discussed. Among these properties is the communication range which may limit which agents are allowed to coordinate with each other. Some of the algorithms create a plan before executing it while other algorithms interleave planning and execution. The latter category of algorithms allow agents to move even though there is not a full solution yet. These methods are known as online algorithms.

One centralized method called Operator Decomposition (OD) deals with the intractability of the problem by considering the possible moves of each agent separately [29, 30]. Instead of taking the Cartesian product of the agents' state spaces it assigns actions to agents individually. This leads to two different kind of states: in standard states no agent has been assigned an action; in intermediate states some of the agents have been assigned an action. When all agents are assigned an action it results in a new standard state. Because intermediate states are considered individually the algorithm is less likely to continue searching the intermediate states that result in longer paths and thus fewer states are generated. The result of this is that the branching factor becomes $(b + 1)$ instead of $(b + 1)^k$. However the depth of the solution in the search tree grows with a linear factor k. This trade-off makes finding a solution with an algorithm like A* more tractable. OD is a complete and optimal algorithm, meaning that it will always find a solution if one exists and it will find the best solution.

On its own OD is not always very efficient so an additional algorithm called Independence Detection (ID) was introduced [29]. Before planning k groups are created, one for each agent, each agent is then placed in its respective group. Each group makes a plan without considering the other groups. When the paths for two groups conflict then each group in turn is tasked with finding a new set of conflict free paths. The groups have to avoid conflicts with each other during this replanning. If both groups fail to resolve the conflict then the groups are merged and a new plan is formed for the new merged group using OD. This process is repeated until a set of conflict free paths for all agents has been found. Combining OD and ID yields an algorithm that has the completeness and optimality benefits of OD while also gaining an increase in speed. Several variants on ID+OD have been proposed, leading to the Optimal Anytime algorithm [30] which will quickly find a solution and can then spend more time on improving the solution. Because ID is an extension that can be applied to any cooperative pathfinding algorithm OD+ID is still complete. Although there is implicit priority in which order the agents are assigned actions. This has no influence on the ability to find a solution or the quality of the solution. The plan is completed before agents start executing it so there is no need to update the plan during execution.

Another centralized method is called the Increasing Cost Tree Search (ICTS) which is a two-fold search method [27]. It consists of a high-level search on an Increasing Cost Tree (ICT) which has a root node that contains the cost of the optimal path for each individual agent. Each child node increases the path cost for a different agent by one. So each level in the tree increases the sum of the path costs by one. This tree is searched using breadth-first search. When a node in the ICT is expanded a low-level search is invoked. This low level search generates all possible paths for all agents that are equal to the cost in the current ICT node. It will then try to find a conflict free combination of these paths. If such a set of paths exists then the algorithm is done. Otherwise the high-level search will continue to the next node in the ICT. Pruning can be used to decrease the amount of duplicate nodes in the ICT. It is possible to use ID with ICTS as well. The ICTS is a complete algorithm like OD+ID. It is faster than OD+ID in situations when the number of agents relative to the number of nodes is high.

The above algorithms both fall into the centralized category of algorithms. These methods can become very slow because of the state-space explosion. Decoupled methods reduce the required calculation time by considering each agent separately. They generally use the same three step approach:

1. Find optimal paths for each agent independent of each other.
2. Impose a hierarchy on the agents, often this is done by assigning them a unique priority.
3. Make new plans for all the agents. This time an agent has to consider all agents with a higher priority as a moving obstacle. Agents with a lower priority are ignored.

This often leads to a set of conflict free plans. Finding the optimal priority ordering is a combinatorial problem [1]. A common algorithm of assigning priorities first calculates a dependence graph based on the paths found in the first step. Then priorities can be assigned such that agents have a priority that is higher than that of agents that may block them. Circular dependencies may mean that multiple priority orderings have to be evaluated. The total costh of the final solution depends highly on the priority ordering employed. Some of the possible priority orderings may not even lead to a solution. This category of algorithms is not complete because it may be the case that none of the possible priority orderings lead to a solution while a solution to the problem does exist.

Most proposals for decoupled methods don't mention whether a central processor must make the plans for all agents or whether the agents can do it themselves. Determining the priority ordering is often centralized since a single processor needs to determine all dependencies [1]. One method called Asynchronous Decentralized Prioritized Planning (ADPP) [5] exploits the inherent parallelism of a multi-robot team during the planning stages. The algorithm allows agents to make their individual plans. After an agent has found a path it will notify all agents with a lower priority of its (new) path. These lower priority agents will then update their plans if conflicts arise. They will in turn notify lower priority agents of their new plan. These agents will then update their plans etc. The benefit of this method is that agents can make a new plan as soon as any one higher priority agent has send a conflicting plan. There is no need for agents to wait for each other to finish their plans. This means that agents can plan simultaneously and that some agents may finish planning before higher priority agents if their paths are conflict free.

Windowed Hierarchical Cooperative A* (WHCA*) is a decoupled algorithm that has been very successful in the video-game industry [28, 2]. It uses a reservation table to denote where agents plan to be and thus prevent other agents from entering the same space at the same time. It requires that agents have been assigned a priority ordering in which they plan so that they can take each other's reservations into account. The amount of computation required depends on the quality of the heuristic used during A* planning. Hierarchical Cooperative A* (HCA*) uses an abstraction of the search space to obtain perfect distance estimates. The reservation table and time dimension are ignored for this abstract space so that the heuristic distance is the same as an agent's optimal path. Agents still use the reservation table to find the conflict free paths. The search by the above algorithm can be limited by using a window. The reservation table is only used in the window and the rest of the path is planned using the same abstract space as HCA*. This effectively ignores the other agent's actions outside of the window. The window is moved at regular intervals and the agent's plan is updated when this happens. When the window moves the priority ordering is recalculated so that the agents have no fixed hierarchy. The priority ordering thus varies based on the current window. Computation is spread out over the time it takes for agents to get to their destination. There is no need to calculate the entire path before execution, instead they can be updated regularly during execution. Agents still ensure that they take the most optimal path to their destination by consulting the abstract space during planning. Usually with decoupled algorithms agents will stop cooperating when they their destination because they have reached their individual goal. This can block other agents from reaching their respective goals. WHCA* solves this by forcing agents to keep planning and coordinating for the length of the window even if the agent has already reached its goal.

The window of WHCA* limits the size of the reservation table that the agents use. In turn this limits the communication range of the agents to the size of the window. Agents share their reservation table with the agents that fall within their window. The other algorithms discussed do not include a limit on the communication range, so far only WHCA* does. Instead those algorithms allow (and often require) all agents to communicate with each other to find a solution. Centralised algorithms use a single processor to find the solution. This means that all agents communicate indirectly with each other through the central processor.

One model of completely decentralized cooperative pathfinding called DM-RCP has been proposed by [35]. Agents move towards their destination and only communicate with other agents that are at most two grid cells away. They can give each other commands like move out of the way, follow me, wait etc. Agents are altruistic which means that they are willing to make concessions during conflicts even if that means that they will be at a disadvantage. Agents use various strategies to deal with different conflict situations. Because of the limited communication range and the various strategies employed the agents often need to recalculate the optimal path to their destination during the execution of their old plan. This approach works well. It requires slightly less computation time than OD+ID and on average the agents only need two thirds of the number of movement steps to reach their goal positions. Although completeness is not discussed the algorithm is based on decoupled methods which are generally not complete. Some of the conflict resolving strategies used by the agents are able to solve situations in which other decoupled methods would not find a solution. Because agents only communicate in a limited range there is no indication whether agents will have conflicts at a later point in time. This lack of a global overview means that agents must include strategies to resolve deadlocks when they occur. There is no way to prevent deadlocks from happening.

Another method that doesn't use a central processor is Distributed Multi-agent Path Planning (DiMPP) [7]. This is a distributed algorithm that is complete, it is guaranteed that it will find a solution. To find a solution all agents are only allowed to communicate in a unidirectional ring: agent a_i receives messages from a_{i-1} and will send messages to a_{i+1}. Counting is modulo n so agent a_n will send its messages to a_1. Sending and receiving messages is done by all agents at the same time. The algorithm finds a solution by evaluating different priority orders. Naively doing so would require the algorithm to evaluate $n!$ priority schemes for n agents. Instead of this naive search the algorithm will only evaluate the orderings

$$\langle a_1, a_2, \ldots, a_{k-1}, a_k \rangle$$
$$\langle a_2, a_3, \ldots, a_k, a_1 \rangle$$
$$\vdots$$
$$\langle a_k, a_1, \ldots a_{k-2}, a_{k-1} \rangle$$

The algorithm now only has to evaluate n orderings instead of all possible $n!$ permutations. The algorithm finds the priority ordering by letting a_1 find its optimal path. It will then send its path to a_2 which will find an optimal path that does not conflict with the path of a_1. After this a_2 will send the global plan (the paths from a_1 and a_2) to a_3. This process of calculating the optimal path for an agent considering the constraints imposed by the paths of the algorithm continues around the ring. If an agent a_i is not able to find a path that has no conflicts with the paths that are already in the global plan then it will reset the global plan to contain no paths. It will now start this procedure again by calculating an optimal path to its destination and putting this as the only path in the global plan and passing the global plan on to a_{i+1}. When an agent a_j receives a global plan in which it already has a path then it knows that all agents have found a conflict free path and the algorithm has found a solution to the problem. In the case that all agents have reset the global plan but no agent ever receives a global plan that includes a path for itself then the algorithm has failed to find a solution. DiMPP has been proven to be a complete algorithm, it will evaluate all n priority orderings which is sufficient to find a solution if one exists. Proof for the completeness of the algorithm are given in [7, subsection 5.1]. The main idea is that an ordering that starts with a_1 will never lead to a solution if any agent is not able to find a conflict free path, it doesn't matter which agents will always have conflicting paths. So when the algorithm evaluates

$$\langle a_1, a_2, \ldots, a_{k-1}, a_k \rangle$$

and fails to find a solution it will not have to consider the $(n-1)!$ other orderings where a_1 has the highest priority. The algorithm requires no central processor but it does not fully exploit the distributed nature of multi-agent systems. Because agent a_{i+1} has to wait for a_i to finish planning there is a dependency between agents that means that they will have to wait until other agents finish their calculations. This algorithm is also not online like most decentralized algorithms because the global plan will be constructed before it is executed.

Optimal Reciprocal Collision Avoidance (ORCA) [31] is a decentralised cooperative pathfinding algorithm that requires no communication between agents. ORCA firs quite well with human behaviour and is most often used in crowd simulation. The algorithm requires that all agents use the same method of collision avoidance. Agents observe each other's position and velocity and use that to construct a velocity obstacle (VO) to predict where the agent goes in the next τ seconds. VOs can also be used to describe the static objects in the environment. An agent will calculate the collision-avoiding velocities that prevent the agents colliding within τ seconds. Multiple VOs can be combined to limit the possible collision-avoiding velocities even further to prevent colliding with multiple agents. ORCA assumes that all agents use the same method of avoiding collisions. Because agents only observe the positions and velocities of nearby agents the algorithm is purely reactive, it requires no communication between agents. Congestions are possible and become common when there are many agents moving in different directions. It can be used together with a global planning algorithm that will determine what the preferred direction for the agent is. ORCA will try to match this as closely as possible. Calculating VOs is so computationally inexpensive that the algorithm can handle hundreds or even thousands of agents in real-time. Most other cooperative pathfinding algorithms are not able to calculate paths for such large numbers of agents in real-time.

3.2 Computational Argumentation

Multi-agent pathfinding can be seen as an instance of a resource sharing problem. From this perspective a conflict occurs when two agents try to access the same resource at the same time. One way of dealing with this resource sharing dispute is by constructing an argument with the goal of determining which agent gets to access the resource at what time. Argumentation has long been studied by philosophers, and in recent decades it has also been extensively researched in the field of Artificial Intelligence as well. In AI it has been studied in the fields of legal argumentation (AI & Law), defeasible reasoning and multi-agent systems. One pillar of argumentation is non-monotonic logic. A logic is non-monotonic when a conclusion that follows based on the premises does not necessarily hold any more when additional premises are added [32, 21, 26]. A classic example of this is that birds can fly, so when you see a bird you assume that it can fly. However when you are told that the bird is a penguin and that penguins can't fly then you will no longer conclude that the bird can fly. A argument is defeasible when it can be defeated by other arguments, in the previous example the fact that the bird can fly is defeasible.

Pollock distinguishes two different types of defeating arguments [23]. *Rebutting defeaters* attack an argument directly and give a reason for an opposite argument. *Undercutting defeaters* do not attack an argument directly. Instead they attack the relation between an argument and its support. The standard example given by Pollock is about an object that looks red: "The ball looks red to John" is a support for John to believe that the ball is red, but there may be a red light shining on the ball. This is a undercutting defeater because it does not attack the conclusion directly, instead it attacks the relation between the observation and the conclusion that the ball is red. After all a white object with a red light shining on it will also look red. Other researchers have formulated additional forms of defeaters, but they can be distilled into three main forms [32]:

Undermining defeaters attack the premises or assumptions of an argument.

Undercutting defeaters attack the connection between a set of reasons and the conclusion in an argument.

Rebutting defeaters raise an argument in favour of an opposite conclusion, thereby attacking an argument.

3.2.1 Dialogues

Multiple agents can have an argument through a dialogue. Walton and Krabbe [33] proposed a typology of the main dialogues that humans partake in. They distinguish six main types of dialogues. It should be noted that the list of dialogue types is not exhaustive. In *information seeking* dialogues some of the participating agents aim to gather information from another agent that knows the answer. In *inquiry* dialogues a group of agents collectively seeks an answer to a question to which none of the participating agents knows the answer on its own. *Deliberation* dialogues are about what course of action to take in a given situation. A *persuasion* dialogue occurs when an agent tries to convince one or multiple other agents of its position. It is successful when the other agent(s) adopt its position. Participants of *negotiation* dialogues try to find a division of a scarce resource that all agents can be satisfied with. Finally *eristic* dialogues are a verbal substitute for fighting. Note that during most human dialogues there can (temporarily) be switched between these types.

Dialogues are often analysed in a game-theoretic sense. The utterances that agents can make are analogous to the moves in a game. Which utterances are appropriate at each moment is defined by the rules of the game. Most of the research into dialogues follows this approach [25, 24]. Most dialogue systems have a two language set-up. The first is the topic language which is about what agents are discussing and is typically a formal logic. It defines the context of the dialogue. The second language is the communication language which specifies which utterances can be made, what effects they have and the rules of outcome.

This latter language is at the core of dialogue games. Most dialogue systems have the following syntax in common [25, 24, 20].

Commencement rules Rules that concern when and how a dialogue can start and what its context is.

Locution rules Which utterances are permitted are known as the locution rules. They may also define when an utterance is obligatory. Common locutions include asserting propositions, questioning or contesting assertions and justifying previous assertions after they have been questioned.

Commitments Some locutions incur commitments on an agent which are subsequently put into the agent's commitment store. A dialogue system may limit which utterances an agent can make based on what is in its commitment store.

Speaker order Most dialogue systems specify an order in which agents can speak, this can range from agents alternating turns to each agent being allowed to make an utterance at any time.

Outcome rules These determine what the outcome of the dialogue is. Some systems define an outcome but allow the dialogue to continue so that it can arrive at a different outcome at a later point in time.

One model of deliberation dialogues is presented in [19], it is also known as the MHP model. It consists of eight stages. It starts with an **Open** stage and ends with a **Close** stage. The other stages form what is called the argumentation phase. Each of those stages can occur multiple times during a dialogue as long as they occur following the rules of the dialogue game. During the dialogue agents will collect the preferences, goals and other constraints that need to be considered. Agents will then propose common plans of action. When multiple plans have been proposed agents can specify which they prefer. In one stage an agent can recommend a plan after which all agents will vote for that plan. The dialogue requires unanimity before the recommended plan is adopted but it allows for any voting mechanism to pick the most preferred plan among many. By gathering the requirements of all agents during the dialogue their local views combine into a single global view that can be used to create a plan. One variant called TeamLog [10] requires fewer stages. Besides the opening and closing stages there are only a proposal and evaluation stage. During these stages agents can still put arguments for or against proposals forward. The TeamLog model has the same expressiveness as the MHP model.

There are also some problems with the MHP model when modelling deliberation dialogues. The model does not have an easy method of integrating additional information into the deliberative process. It also doesn't have a method of dealing with failures to find a course of action. The closing stage can only be reached when the agents have settled on a specific plan. It may be the case that it is not possible to find a satisfactory solution. This makes it impossible for the dialogue to reach the closing stage. These two shortcomings are raised and addressed by [34]. The problem of integrating additional information into the dialogue is addressed by adding a knowledge base that is specific to the dialogue which is initially filled with information in the opening phase. It is possible to extend the knowledge base in the information seeking stage of the dialogue. The extended model lists ten criteria for when a dialogue can be closed. Some of these reasons are: all proposals were discussed, the quality of the arguments in support/attack of a proposal, whether agents followed procedural rules, and the accuracy of the knowledge base.

Other approaches to distributed deliberation dialogues in cooperative multi-agent systems based on DeLP and MAPOP have been proposed [12, 22]. In these systems agents make partial ordered plan proposals and argue for or against them. Agents share information that they have about the world and their objectives. During dialogues agents share their plans and they are allowed to argue for or against a plan, this can be on the level of individual actions. The dialogues prescribe a turn order for agents such that during each round of argumentation each agent gets the opportunity to submit plans, threats or arguments. During each round the global plan will become more refined. The agents collectively search for the most appropriate plan with an algorithm analogous to A*.

3.3 Multi-agent coordination

Argumentation can be used to allow coordination between agents by letting them deliberate in a dialogue. Cooperative pathfinding is a particular instance of a coordination problem. Before we can combine cooperative pathfinding with practical reasoning we have to consider the argumentative method of building plans for a single agent that was introduced by Pollock [23]. An agent starts out by making a global plan consisting only of coarse steps. This saves computation time and it defers planning specific actions to a later time when more information about the problem becomes available. When the agent reaches a step in a plan that is not concrete enough yet it will start constructing a sub-plan for that step. It may also start sub-planning this when another planning process depends on it. This is done in multiple levels leading to a hierarchical plan. The lowest level consists of basic actions that are inherent in the agent (like lifting an arm). At the same time the agent also keeps track of whether it is still possible to execute the future steps in the plan. The agent will have to adapt its plans once it notices that it is not possible to execute the remainder of the plan any more for any reason. This allows an agent to adapt to a changing environment and changing desires. Although this design focusses on planning actions for a single agent it can easily be extended to planning for groups of agents.

Coordination in a multi-agent system can be done through Partial Global Planning (PGP) [11, 8, 36, pp. 202–204]. The goal is to let agents cooperate without any one of them formulating a global plan. Instead agents will coordinate with other agents only when they need to. This leads to the construction of many small local plans which can be communicated to other agents as well. The result is that eventually there will be a global plan that covers all agents. The agents themselves will only know the part of the global plan that is relevant to them. The global plan is implied by these partial global plans. Key to partial global planning is that no agent needs to know the global plan, it only needs to know which parts of the plan it is affected by. This approach is similar to that of decoupled cooperative pathfinding because they use a similar planning structure. Partial global planning starts out with letting each agent make their individual goals. Next agents communicate information on where plans interact. Finally they will alter their plans such that their actions are better coordinated and there are no negative influences. Generalized PGP [8] extends this with real-time planning, negotiation, and coordination relationships between goals. This allows the framework to be used in settings other than the multi-sensor network that PGP was originally developed for.

Continual Planning [3, 4] aims to achieve coordination in a multi-agent setting where the environment can be partially observable and is highly dynamic. Here plan creation and execution are interleaved so that agents are better able to respond to changes in their environment. This is similar to Pollock's OSCAR [23] but Continual Planning specifies when switching between planning and executing should happen and it is designed to work in a multi-agent system. *Assertions* are used as preconditions to switch between planning and execution. During the planning phase an agent will postpone creating a plan for a sub-problem and create an assertion instead. The agent can start executing the plan when it has created these assertions. When the assertion is satisfied then the agent will stop executing and the planner will resume planning and find a way to achieve the sub-goal for which planning was originally postponed. Agents can also ask each other to achieve certain goals or execute actions. Often agents will request of another agent to reach a sub-goal instead of executing a multi-step plan. The agent can then determine its own plan to achieve this new sub-goal and its other goals. This allows for flexibility in cooperation as agents are able to plan according to other constraints that may have been imposed. Continual Planning has been applied to the cooperative pathfinding problem. One of the main finds was that a full view of the problem does not necessarily lead to a better solution. Agents with a limited sensing and communication range are often able to find a solution in the same time while the length of their paths is about equal. This is attributed to the difficulty of finding a plan with full observability is often hard to do and slow while finding a partial plan, executing it and finding a new partial plan when new conflicts arise is faster. This comes at the trade-off that agents may get stuck during the execution and reach a state in which no plan can be found that successfully solves the cooperative pathfinding problem. These findings are similar to those of using a window to restrict cooperation to a limited temporal range in WHCA* [28].

4 Family of algorithms

Decoupled algorithms are able to solve cooperative pathfinding problems while requiring only minimal computational resources. The agents calculate paths to their destination individually so this is inherently distributed. A hierarchy is imposed on the agents by assigning them a priority order. This order allows agents to avoid conflicts without a central processor making a plan. However a central processor is still needed to determine the priority order. A central processor is used to find dependencies between agents to determine possible priority orderings that can be used to solve the problem [16, 1]. This means that decoupled methods are mostly distributed with a single centralized bottleneck. All agents will have to halt calculating a solution while the priority ordering is determined by the central processor. The central processor in its turn has to wait for all agents to calculate and communicate their optimal paths before it is able to calculate the priority order.

To overcome this bottleneck the calculation of the priority ordering can also become distributed. The decoupled method can be altered to allow for this.

The first step where each agent plans its optimal path without regard for the other agents remains the same. Next agents share the paths that they found with each other. Each agent can now determine where its path and the paths of other agents have conflicting moves that would lead to a collision. The agents will then be able to solve the conflicts that occur without having to wait for slower agents to calculate and communicate their optimal paths. To solve conflicts agents will start a dialogue where possible solutions are proposed, evaluated and adapted. The proposals made consist of a priority order for the agents involved in the conflict. Agents will only need to solve the first conflict that occurs in their path because solving it may have the side-effect of solving or causing later conflicts. After a conflict has been successfully solved then the agents involved can work on solving the next conflict. Below are the details of three different versions of this algorithm. Each version of the algorithm has some improvements over the previous version.

Three algorithms build upon a more general search algorithm Cooperative A* (CA*). This algorithm is a variation on A* [13] that allows teams of agents to cooperate. Each agent searches for a path individually, but they can take each other's actions into account. The algorithm searches both the space and time dimensions to ensure that paths are conflict free. To be able to do this an agent needs to know the paths of the other agents that can potentially conflict with its own path. The algorithm works like regular A* with the addition that it has to consider the moves that other agents make. It does this by not taking actions that would cause a conflict with the path of an agent of higher priority. This results in a path which leads the agent to its destination and it does not collide with other agents during the execution of this plan. Because CA* is based on A* it will find the optimal path which does so.

4.1 Partial Cooperative A* (PCA*)

The heart of the algorithm is the conflict resolution step. The most straightforward approach to solving conflicts is by going through all possible agent orderings. An ordering determines which agent has priority over another agent. The ordering $a_1 > a_2$ indicates that a_1 can plan freely while a_2 has to consider a_1 as a moving obstacle. Usually decoupled methods use a permutation of the priority ordering $a_1 > a_2 > \ldots > a_k$ that all agents have to adhere to. Our algorithm Partial Cooperative A* (PCA*) does this for smaller groups of agents, it is outlined in Algorithm 1. Initially agents find their optimal paths without considering the presence of other agents (line 1 and line 2) and communicate the result with each other (line 3). The function FindPath() finds a path for an agent with the constraints of the priority orderings given as its sole argument. CommunicatePath() sends this path to all other agents so that they can find conflicts. HasConflict() will return true when an agent has conflicts along its path or false otherwise. When agents detect that there is a conflict in their individual plans then they will try to find a priority ordering between them that will solve the conflict. Agent will always try to find a solution to the conflict that is closest to their current position first (line 5) given by the function EarliestConflict(). This is because the solution to earlier occurring conflicts may have the side-effect of solving or creating later conflicts. There is no need to waste computational resources on solving a conflict that will be solved by implication when an earlier occurring conflict is solved.

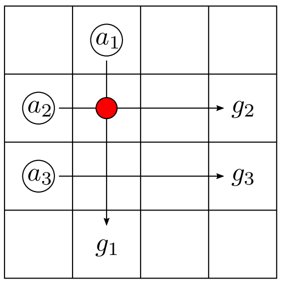
Algorithm 1 Partial Cooperative A*

```
1: $Permanent \leftarrow \emptyset$
2: $Path \leftarrow$ FindPath($Permanent$)
3: CommunicatePath($Path$)
4: while HasConflict() do
5:     $conflict \leftarrow$ EarliestConflict()
6:     $Orderings \leftarrow$ PriorityOrderings($conflict$)
7:     $Cost \leftarrow \emptyset$
8:     for all $ordering \in Orderings$ do
9:         $Path \leftarrow$ FindPath($Permanent \cup ordering$)
10:        $Cost[ordering] \leftarrow$ PathCost($path$)
11:    end for
12:    $Permanent \leftarrow Permanent \cup \{\arg\min_{ordering} Cost\}$
13:    $Path \leftarrow$ FindPath($Permanent$)
14:    CommunicatePath($Path$)
15: end while
```

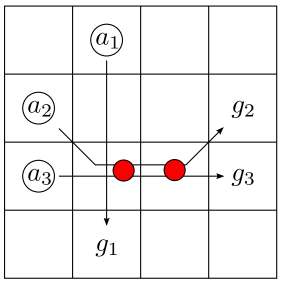
To find the most suitable priority ordering all possible orderings between the agents that are involved in the conflict are evaluated (line 6–11 in Algorithm 1). A conflict with two agents will result in the orderings $a_1 > a_2$ and $a_2 > a_1$ being evaluated. The function PriorityOrderings() will give the set of all possible priority orderings permutations for the agents involved in the conflict. The agents temporarily adapt the first ordering and plan new paths with the constraints it introduces (line 9). The agents measure the length of the paths that they found using the PathCost() function (line 10). They do this for each priority ordering that is possible. The priority ordering with minimal increase in sum of path lengths is permanently adapted by the agents (line 12). A new path is calculated and communicated with all other agents (line 13 and line 14). Because the solution with the lowest sum path length is used there is no consideration for the effects that the solution has on conflicts that occur later.

Only the agents that occur in a priority ordering adapt it. This means that an agent a_3 does not know about the ordering that agents a_1 and a_2 have settled on, say that they picked $a_2 > a_1$. When a_3 and a_1 have a conflict then PCA* will also have to find a solution for it. If they settle on the solution $a_1 > a_3$ then only a_1 knows $a_2 > a_1 > a_3$, the other two agents only know their partial priorities. In this case a_1 holds all information to obtain the global priority ordering. Often it is not the case that a single agent knows the full global priority ordering. The global ordering is implied by the local partial orderings that the agents do know about. This is similar to how plans are constructed in partial global planning where no agent knows what the global plan is either.

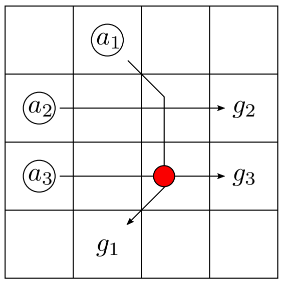
The orderings that are found do not need to be unambiguous, if a_1 and a_3 had used the solution $a_3 > a_1$ then a_1 would have orderings $a_2 > a_1$ and $a_3 > a_1$ but $a_2 > a_3$ or $a_3 > a_2$ is not known. This leaves a_2 and a_3 free to use any priority ordering in the event that they also have a conflict in their paths. This also allows for circular priority orderings, something which conventional decoupled algorithms do not support [1]. This is possible because the circular ordering is implicit in all the partial orderings that individual agents know about. The



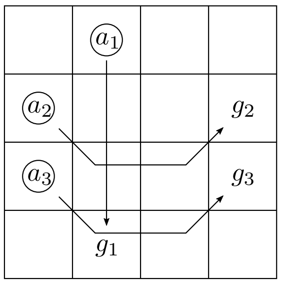
(a) Initial configuration showing $p_{a_1,1}$, $p_{a_2,1}$ and $p_{a_3,1}$.



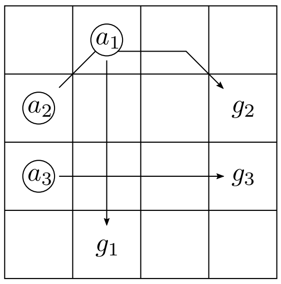
(b) Configuration after priority ordering $a_1 > a_2$. Agent 2 now has path $p_{a_2,2}$.



(c) Configuration after priority ordering $a_2 > a_1$. Agent 1 now has path $p_{a_1,2}$.



(d) Configuration after priority orderings $a_1 > a_2$ and $a_2 > a_3$. Paths shown are $p_{a_1,1}, p_{a_2,2}, p_{a_3,2}$.



(e) Configuration after priority orderings $a_1 > a_2$ and $a_3 > a_2$. Paths shown are $p_{a_1,1}, p_{a_2,3}, p_{a_3,1}$.

Figure 3: Five stages of resolving a conflict using PCA*. Agents are circles inscribed by a_i, their respective goals are g_i. Paths that are followed are indicated by the arrows. The circles indicate where agents have conflicting moves in their paths.

implied global priority also doesn't need to be complete: not every agent needs to be present in it. This is easiest to see when considering only agents a_1 and a_5 in Figure 1. For this example the other agents are not relevant. There is no need to establish an ordering for these two agents since their paths don't intersect. This has the effect of implied Independence Detection [29] because these agents will never have to communicate beyond sharing the paths that they have found with each other. There is no need for them to coordinate because their plans never interact.

4.1.1 Example of conflict resolution

Consider the configuration of agents shown in Figure 3a. It shows a 4×4 grid that contains the agents a_1, a_2, and a_3 with goal positions g_1, g_2, and g_3 respectively. The optimal paths to their destinations are shown as arrows. Agent a_1 has a path that consists of three *south* moves, $p_{a_1,1} = \{\textit{south, south, south}\}$, while both a_2 and a_3 have paths that consist of three consecutive *east* moves, $p_{a_2,1} = p_{a_3,1} = \{\textit{east, east, east}\}$. None of the agents has a *wait* action in their path. After the agents have calculated and shared their optimal paths they find that a_1 and a_2 have a conflict after their first action.

Before discussing the details of how the agents resolve the conflict in this situation some definitions are needed. A proposal where agent a_i has priority over agent a_j is represented as $a_i > a_j$. To resolve this conflict they evaluate the priority ordering proposals $a_1 > a_2$ and $a_2 > a_1$. The situation after adapting $a_1 > a_2$ is shown in Figure 3b, it shows that a_2 has a new path $p_{a_2,2} = \{$*south east, east, north east*$\}$. The situation after adapting $a_2 > a_1$ is shown in Figure 3c, it shows that a_1 has a new path $p_{a_1,2} = \{$*south east, south, south west*$\}$. Both of these paths have an equal length so neither of them is strictly better. In this example ties are broken in favour of the lower numbered agent, so the priority ordering $a_1 > a_2$ is permanently adopted by a_1 and a_2. Agent a_3 does not adapt the priority ordering.

The situation is now as shown in Figure 3b where the paths $p_{a_2,2}$ and $p_{a_3,1}$ have two conflicts. Agent a_1 is not involved in these conflicts because it visits the first conflict location after a_2 and a_3 leave it. To resolve this conflict the agents evaluate the priority orderings $a_2 > a_3$ and $a_3 > a_2$. The situation after adopting $a_2 > a_3$ is shown in Figure 3d, it shows that a_2 still has path $p_{a_1,2}$ because $a_1 > a_2$ still applies while a_3 has now adapted the path $p_{a_3,2} = \{$*south east, east, north east*$\}$. The situation after adopting $a_3 > a_2$ is shown in Figure 3e, it shows that a_2 has a new path $p_{a_2,3} = \{$*north east, east, south east*$\}$ while a_3 has its original path $p_{a_3,1}$. Both of these paths have length 3 so the conflict is settled in favour of the lower numbered agent, a_2 and a_3 permanently adopt the ordering $a_2 > a_3$.

As a result of the above conflict resolution process each agent now holds part of the implied global priority ordering. Agent a_1 knows $a_1 > a_2$, agent a_2 knows $a_1 > a_2$ and $a_2 > a_3$, and a_3 knows $a_2 > a_3$. There is a global priority ordering $a_1 > a_2 > a_3$ which can only be derived by a_2, the other two agents have insufficient knowledge to construct the global priority ordering.

4.2 Dialogue-based Partial Cooperative A* (DPCA*)

PCA* evaluates all possible partial priority orderings for a conflict to obtain the most appropriate solution. Doing so can be computationally expensive even when only a small number of agents need to be considered. These permutations are only evaluated on their increase of solution cost, while they may also have other effects on the global state. Some improvements can be made to PCA* so that it does not exhaustively search all partial priority orderings while also considering their side-effects. Deliberation dialogues can be utilized to achieve this. Agents take part in a dialogue of which the goal is to find a solution to the conflict that works for all agents involved in the conflict. This dialogue consists of several stages that are summarised in Table 2. There is a separate dialogue for each conflict. The agents work through the conflicts in chronological order,

Table 2: Stages of a conflict resolution dialogue.

Stage	Goal	Next stage
Opening	Exchange information	Proposal
Proposal	Make (incomplete) priority proposals	Evaluation
Evaluation	Vote on suitability of proposals	Proposal, Closing
Closing	Permanently adapt best proposal	

Algorithm 2 Dialogue-based Partial Cooperative A* (DPCA*)

```
Require: topic: conflict that is to be solved by the dialogue
 1: $Permanent \leftarrow \emptyset$
 2: $Path \leftarrow FindPath(Permanent)$
 3: $CommunicatePath(Path)$
 4: $Conflicts \leftarrow$ FindConflicts()
 5: for all $conflict \in Conflicts$ do                    ▷ Stage 1: Opening
 6:     if $conflict \neq$ EarliestConflict() then
 7:         PutDialogueOnHold($conflict$)
 8:         continue
 9:     end if
                                                         ▷ Stage 2: proposal
10:     repeat
11:         Propose()
                                                         ▷ Stage 3: evaluation
12:         for all $proposal \in unevaluatedProposals$ do
13:             $path \leftarrow$ FindPath($permanent \cup proposal$)
14:             $vote, expand \leftarrow$ Evaluate($path$)
15:             CastVote($vote$)
16:         end for
17:     until $\neg expand$
                                                         ▷ Stage 4: closing
18:     $permanent \leftarrow permanent \cup \arg\min \sum votes$
19:     $Path \leftarrow$ FindPath($Permanent$)
20:     CommunicatePath($Path$)
21:     $Conflicts \leftarrow$ FindConflicts()
22: end for
```

they solve conflicts that occur early before solving conflicts that occur later. The dialogue replaces lines 6–12 of Algorithm 1. A more complete outline of the dialogue-based algorithm is given in Algorithm 2. There are several additional functions in Algorithm 2 that were not found in Algorithm 1. The function FindConflicts() returns the set of all conflicts that an agent is involved in. PutDialogueOnHold() will tell the other agents in a conflict to wait until the agent is willing to continue. Finally CastVote() allows agents to vote on a proposal. All of these processes will be discussed in the remainder of this section.

Each dialogue starts with an opening stage, during this stage the agents notify each other if they are taking part in any dialogues for conflicts that occur earlier than the current conflict being discussed. If there is such a prior dialogue then the current dialogue will be put on hold until all earlier dialogues are completed, this achieved by lines 6–9 of Algorithm 2. If the conflict of the dialogue is the conflict that occurs the earliest for all involved agents then the dialogue moves on to the proposal stage.

During the proposal stage each agent can enter a new ordering proposal that is to be evaluated. Agents can make only one single proposal during each proposal stage. There can be multiple of these stages during a dialogue so it is possible for agents to make multiple proposals before the dialogue has concluded.

The proposed priority can be partial, if there are three or more agents taking part in the dialogue then $a_1 > a_2 > a_3$ is a valid proposal but $a_1 > a_2, a_3$ is as well. In the latter case a_1 has priority over both a_2 and a_3, but there is no established priority ordering between a_2 and a_3 yet and this may be decided upon later during the dialogue, or during a future dialogue. Agents will always propose that they get a higher priority over the other agents that are part of the conflict. So in a dialogue that involves two agents a_4 and a_5 each agent gets to make a proposal, a_4 will propose $a_4 > a_5$ while a_5 will propose $a_5 > a_4$. Agents also have the option to not make a proposal during this stage.

The third stage is the evaluation stage which is reached when all agents have made a proposal or declined to make one. Each of the new proposals will be evaluated in turn. To evaluate a proposed priority ordering the agents temporarily adapt it and update their plans. During this replanning agents have to take into account the constraints imposed by both the ordering in the proposal, and the ordering imposed by previously solved conflicts. Once an agent has updated its plan then it will cast a vote based on how suitable the proposal is. When an agent finds that it is unable to plan a path to its destination under a certain proposal then it will notify the other agents of this. In this case the proposal is rejected by all agents and not voted on, it can also not be expanded on during an extra proposal stage. If it is not the case that an agent is blocked from reaching its destination then all agents will show their preference by voting on the proposals. Each vote consists of a real number that represents how suitable the proposal is. This number is based on the increase of the length of the path, and whether the new plan solves or causes more conflicts at later time steps. Both of these factors are weighted to result in the final vote. All agents cast a vote on each acceptable proposal. The proposal with the lowest sum of the votes is accepted as the solution to the conflict. The votes of each agents are weighted equally so there is no agent which has a stronger vote.

After all the proposals have been evaluated there is room for agents to claim to want to expand on previous proposals. If an agent does so then the dialogue goes through another proposal and evaluation stage. If no agent wants to make additional proposals the dialogue can be completed in the closing stage. During the closing stage each agent will permanently adapt the priority ordering with the highest sum of votes. The priority ordering in this proposal is always considered when making new proposals and plans during future dialogues. This completes the dialogue, the agents can now work on conflicts that still occur. The entire above process is repeated for all conflicts until they are all solved.

In conflicts that involve three or more agents it is not always the case that a priority ordering will solve the conflict for all agents. In some conflicts involving agents a_1, a_2, a_3 it may be the case that a partial ordering $a_1 > a_2, a_3$ means that a_1 will not have a conflict with a_2 and a_3 any more, but that a_2 and a_3 will still have a conflict at another position and/or time. In this case either agent can make a request for an additional proposal round. During this proposal round the agents can make new proposals or expand on additional proposals. Agents do not need to make proposals so a_1 might not make any new proposals because it already has the highest priority in the proposal $a_1 > a_2, a_3$. On the other hand a_2 and a_3 will propose $a_1 > a_2 > a_3$ and $a_1 > a_3 > a_2$ respectively. After all three agents have entered a proposal or declined to make one the dialogue moves to the evaluation stage again. This time only the new proposals are evaluated. Any duplicate proposals are rejected.

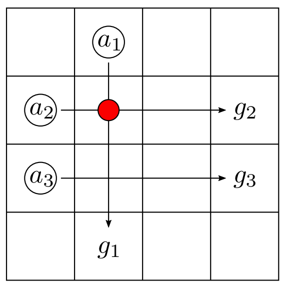
Conflicts that involve more than two agents can be solved in two different ways. The first is to let agents solve the conflict in pairs, this approach is known as Dialogue-based Partial Cooperative A* (DPCA*). In a conflict between the agents a_1, a_2 and a_3 at time t then there would be three dialogues: one between a_1 and a_2, one between a_1 and a_3, and one between a_2 anda_3. Say that a_1 and a_2 are the first to hold a dialogue which finishes with the priority $a_1 > a_2$, meaning that a_1 has priority over a_2. Agent a_2 will have found a path that does not go through the location of the conflict at t. This has the effect of also solving the conflict between a_2 and a_3. Now only a_1 and a_3 still have a conflict at t and they will have to hold a dialogue about which agent gets priority over the other. When this dialogue would end with the priority ordering $a_1 > a_3$ then the multi-agent conflict at t is solved. It may be the case that a_2 and a_3 now have another conflict at a different position and/or time that they will have to resolve. For this conflict there will be at least two dialogues that need to lead to a conclusion, and at most three conflicts if a_2 and a_3 do have a conflict at a different location.

The other approach to solving conflicts in which more than two agents are involved is to have a single dialogue in which all agents participate, this is known as Dialogue-based Partial Cooperative A* Plus (DPCA*+). Having multiple agents in a dialogue will require that the dialogue supports partial priority orderings and that it allows for multiple rounds of making proposals and evaluating them. DPCA* does not need to have this complexity in dialogues. Evaluations are also more complex because agents have to weigh whether a proposal solves the conflict for just a subset of the involved agents. This is effectively a penalty for a proposal that only partially solves the conflict under discussion. DPCA*+ may be more complex than DPCA*, but it also requires fewer dialogues to find a set of conflict free paths and therefore it may be faster than DPCA*.

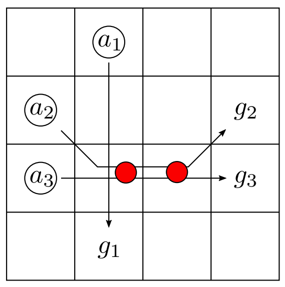
Each time that agents evaluate a proposal they have to compute paths that satisfy the constraints imposed by the priority orderings. This often means that agents have to recompute the same paths when their priority doesn't change between proposals. Te reduce the amount of computation required agents can store the paths that they have calculated. When agents need to calculate a path to evaluate a proposal they can consult their path cache. This allows them to use a path that has been found earlier and use that as a solution. The only restriction is that the cached path does not have a conflict with the paths of agents that have a higher priority. Some cached paths may cause new additional conflicts with lower priority agents. This should not stop agents from using this path because they can solve these conflicts in a later dialogue. Using a cache should reduce the overall amount of time that is required to find a conflict free set of paths for all agents.

4.2.1 Example of dialogue-based conflict resolution

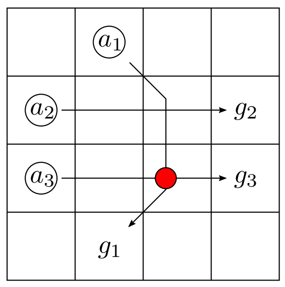
The cooperative pathfinding problem in Figure 3 can also be solved using DPCA*. This process is outlined using Figure 4. The problem and the initial paths are equal. After agents share their initial path they will find where their paths conflict. Agents a_1 and a_2 discover that they collide after their first action. This means that they will have to resolve this conflict to prevent the collision. To allow agents to evaluate proposals to solve the conflict we define



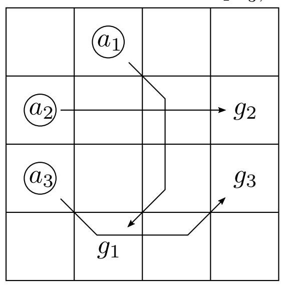
(a) Initial configuration showing $p_{a_1,1}$, $p_{a_2,1}$ and $p_{a_3,1}$.



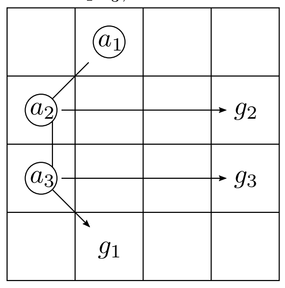
(b) Configuration after proposal $a_1 > a_2$. Shown are paths $p_{a_1,1}$, $p_{a_2,2}$, $p_{a_3,1}$.



(c) Configuration after proposal $a_2 > a_1$. Shown are paths $p_{a_1,2}$, $p_{a_2,1}$, $p_{a_3,1}$.



(d) Configuration after proposal $a_1 > a_3$. Shown are paths $p_{a_1,2}$, $p_{a_2,1}$, $p_{a_3,2}$.



(e) Configuration after proposal $a_3 > a_1$. Shown are paths $p_{a_1,3}$, $p_{a_2,1}$, $p_{a_3,1}$.

Figure 4: Five stages of resolving a conflict using DPCA*. Agents are circles inscribed by a_i, their respective goals are g_i. Paths that are followed are indicated by the arrows. The circles indicate where agents have conflicting moves in their paths.

$\text{vote}_{a_n}(a_i > a_j)$ as the vote of agent a_n on the proposal $a_i > a_j$. This represents how suitable an agent thinks that a particular proposal is. The evaluation is based on two effects which can be weighted differently. The first effect is the difference in path length before and after a proposal has been adopted. The second effect is the change in the number of conflicts that an agent is involved in, this effect is weighted three times heavier than the former effect. Qualitatively this means that $\text{vote}_{a_n}(a_i > a_j) = 1 \cdot \Delta l + 3 \cdot \Delta c$ where Δl is the difference in path cost and Δc is the difference in the number of conflicts. The value of the weights have been arbitrarily picked for demonstration purposes. Their value should be set empirically in an implementation. Agents only communicate their final vote and not how they arrived at it.

Continuing with the example, after the agents have determined that they have a conflict they start a new dialogue. The dialogue starts in its initial stage; the opening. All messages are broadcast to all agents in the conflict dialogue, in this case a_1 and a_2:

a_1: no earlier conflicts

a_2: no earlier conflicts

Because both agents don't have any conflicts that occur at an earlier time (this is the first time step) the dialogue can move on to the proposal stage. Both agents make a proposal in which they go first:

a_1: propose $a_1 > a_2$

a_2: propose $a_2 > a_1$

Both agents have made a proposal to resolve the conflict. The dialogue can move to the evaluation stage. Proposals are evaluated in numerical order of the agent that made the proposal so a_1's proposal will be evaluated first. In this proposal a_1 has the highest priority so it doesn't need to update its plan and its path remains $p_{a_1,1}$. It updates a_2 of the fact that its path hasn't changed and still consists of three consecutive *south* actions:

a_1: new path

$p_{a_1,1} = \{\textit{south, south, south}\}$

Agent a_2 does have to yield to a_1 so it will have to consider possible conflicts that arise with $p_{a_1,1}$ and plan around them. It finds a new path $p_{a_2,2} = \{\textit{south east, east, north east}\}$ which is shown in Figure 4b. After finding the new path a_2 will evaluate its quality so that it can send its vote to a_1. After adopting the proposal a_2 now has two conflicts with a_3 so $\Delta c = 1$. There is no difference in the length of the paths before and after temporarily adopting the constraints of the proposal so $\Delta l = 0$. This means that a_2 can send the vote to a_1. It will send its new path along with the evaluation:

a_2: new path

$p_{a_2,2} = \{\textit{south east, east, north east}\}$

a_2: $\text{vote}_{a_2}(a_1 > a_2) = 3$

Now that a_1 knows the new path of agent a_2 after adopting to $a_1 > a_2$ it can also vote on the proposal. This happens in a similar vein as a_2's evaluation. Agent a_1 has no more conflicts while it had one previous conflict so $\Delta c = -1$. The vote for this proposal can then be cast:

a_1: $\text{vote}_{a_1}(a_1 > a_2) = -3$

Now that this proposal has been evaluated by both agents they can find the sum score of the proposal which is $\text{eval}(a_2 > a_1) = \text{eval}(a_1, a_1 > a_2) + \text{eval}(a_2, a_1 > a_2) = 0$. Next the agents can evaluate $a_2 > a_1$. When agent a_1 goes to evaluate this conflict if finds that it will have to wait for a_2 to send a new path because a_2 currently has a temporary path. Agent a_2 can use the path $p_{a_2,1}$ which is stored in its cache, it sends this information to a_1:

a_2: new path

$p_{a_2,1} = \{\textit{east, east, east}\}$

Next a_1 can evaluate the proposal. First it needs to make a new plan that does not conflict with the path that a_2 has just sent. It finds the path $p_{a_1,2} = \{\textit{south east, south, south west}\}$. This new situation is shown in Figure 4c. Agent a_1 can immediately evaluate it, the lengths of $p_{a_1,1}$ and $p_{a_1,2}$ are equal and there is an equal amount of conflicts. The new path and the evaluation are communicated with a_2:

a_1: new path

$p_{a_1,2} = \{\textit{south east, south, south west}\}$

a_1: $\text{vote}_{a_1}(a_2 > a_1) = 0$

Now that a_2 knows a_1's new plan it can also send its evaluation which has one fewer conflict than its original path:

a_2: $\text{vote}_{a_2}(a_2 > a1) = -3$

Now both agents have evaluated the proposal $a_2 > a_1$ they can find the sum of the evaluations which is $\text{vote}(a_2 > a_1) = \text{vote}_{a_1}(a_2 > a_1) + \text{vote}_{a_2} = -3$. All proposals have been evaluated so the agents can notify each other if they want

to make further proposals:
a_1: no further proposals
a_2: no further proposals

Neither agent wants to make more priority ordering proposals. This is because there are no more possible priority orderings to make with these two agents. The dialogue can then move to the closing stage. In the closing stage agents will pick the best proposal, in this case that is the proposal with minimal sum score which is the proposal $a_2 > a_1$ (Figure 4c). To adapt this proposal agents will need to use the respective paths that they used during the evaluation of the proposal. So a_1's path will be $p_{a_1,2}$ and a_2's path will be $p_{a_2,1}$. They also need to keep track of which agents have a higher priority than themselves. Agent a_1 must now store that a_2 has a higher priority while a_2 does not have to store anything. Both agents also communicate their new path with all other agents, in Figure 4 this is only agent a_3.

After the new path have been evaluated a_1 and a_3 notice that they have a conflict. Their dialogue is as follows:

Opening stage
a_1: no earlier conflicts
a_3: no earlier conflicts
Proposal stage
a_1: propose $a_1 > a_3$
a_3: propose $a_3 > a_1$
Evaluation stage, starting with $a_1 > a_3$, new paths are shown in Figure 4d
a_1: new path
$p_{a_1,2} = \{$*south east, south, south west*$\}$
a_3: new path
$p_{a_3,2} = \{$*south east, east, north east*$\}$
a_3: $\text{vote}_{a_3}(a_1 > a_3) = -3$
a_1: $\text{vote}_{a_1}(a_1 > a_3) = -3$
Evaluation of $a_3 > a_1$, new paths are shown in Figure 4e
a_3: new path
$p_{a_3,1} = \{$*east, east, east*$\}$
a_1: new path
$p_{a_1,3} = \{$*south west, south, south east*$\}$
a_1: $\text{vote}_{a_1}(a_3 > a_1) = -3$
a_3: $\text{vote}_{a_3}(a_3 > a_1) = -3$

Both proposals have a sum score of -6 so we break the tie in favour of a_1, $a_1 > a_3$ is permanently adapted by both agents. The final paths $p_{a_1,2}$ and $p_{a_3,2}$ are communicated with a_2 and there are no more conflicts. A valid solution as been found: $a_1 > a_3$ and $a_2 > a_1$.

DPCA* finds a different solution than PCA* because it takes the number of conflicts in a partial solution into account. Because of this it finds a different solution to the conflict between a_1 and a_2. DPCA* also added more transparency to the process by showing why agents picked a certain solution to a conflict. In contrast PCA* is opaque, the result was mainly determined by how ties between equal priority orderings are broken.

4.3 Windowed Dialogue-based Partial Cooperative A* (WD-PCA*)

DPCA* and DPCA*+ have to compute the entire solution before it can be executed. Planning and execution both take time without requiring the same resources making it possible to do them at the same time. This means that the plan can be executed while it is still being constructed. One way of doing this is by applying a window to restrict how far away from an agent's location DPCA* will be used to solve conflicts. A window w determines that agents will use the above algorithm to solve all conflicts that occur within w time steps from their current position. Agents will not cooperate past the boundary of the window, solving conflicts that happen beyond that border is deferred to a later point in time. Periodically the agents will update the conflicts that occur within their window and solve them. Because agents only coordinate in the window it is not necessary for them to plan a path past the window boundary as well. Instead agents can plan a path for the next w time steps so they get closer to their goal. To achieve this the graph can be changed so that the nodes at the window boundary connect directly to the goal node. This can be achieved by changing the cost function defined in Section 2 between adjacent nodes P and Q [28]

$$\textsc{cost}(\text{P,Q}) = \begin{cases} 0 & \text{if } P = Q = G, t < w \\ \textsc{HeuristicDistance}(\text{P,G}) & \text{if } t = w \\ 1 & \text{otherwise} \end{cases}$$

where HeuristicDistance is a function that returns the cost of the shortest path between P and G as if there are no other agents on the graph.

Using the window spreads out the computation over the course of execution, but it has other benefits as well. Exchanging optimal paths between agents after the first step may take a long time in large multi-agent systems. By limiting the search using a window agents only need to coordinate with a limited number of other agents. This reduces the initial planning time, as well as for each subsequent window. Agents that are never in each other's window will never have to communicate with each other, saving a lot of unnecessary communication and conflict detection overhead. Windowing the search also has benefits in systems where agents can change their destination during execution. Instead of recalculating the entire plan when this happens, only agents within the window of the agent changing destination have to update their plan. Agents that are not affected by the change in destination do not have to update their plan. When there would be no window all agents would have to recalculate and solve all conflicts again, even if they would not need to update their plan, leading to wasted computational resources.

An online algorithm like WDPCA* has the implied effect of forcing agents to abide the consequences of interleaving planning and execution. It is not desirable for agents to backtrack long distances to solve a conflict that occurs late in execution because it would drastically increase the solution cost. Instead agents will have to deal with the consequences when they occur. This fits with the philosophy of partial global planning where a solution is incrementally arrived at. By employing a window agents will set parts of their plan in stone while keeping future actions flexible. This is similar to how partial global planning enforces agreed upon coordination but allows agents flexibility in the details of their plan.

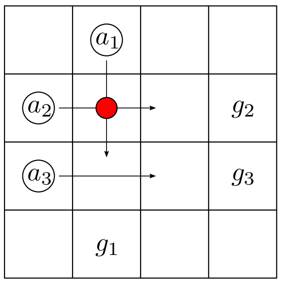
4.3.1 Example of dialogue-based conflict resolution

The same situation as in Figure 3 and Figure 4 can also be solved by WDPCA*. This process is shown in Figure 5. For this example the window size is set to two ($w = 2$), agents always make plans of two steps. Initially agents a_1 and a_2 have a conflict, their dialogue:

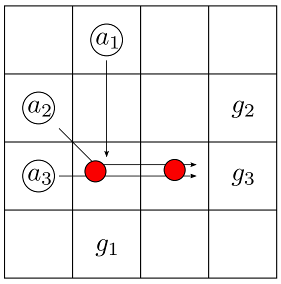
Opening stage
a_1: no earlier conflict
a_2: no earlier conflict
Proposal stage
a_1: propose $a_1 > a_2$
a_2: propose $a_2 > a_1$
Evaluation stage, starting with $a_1 > a_2$, new paths are shown in Figure 5b
a_1: new path
$p_{a_1,1} = \{\textit{south, south}\}$
a_2: new path
$p_{a_2,2} = \{\textit{south east, east}\}$
a_2: vote$_{a_2}$ $a_1 > a_2 = 3$
a_1: vote$_{a_1}$ $a_1 > a_2 = -3$
Evaluation of $a_2 > a_1$, new paths are shown in Figure 5c
a_2: new path
$p_{a_2,1} = \{\textit{east, east}\}$
a_1: new path
$p_{a_1,2} = \{\textit{south east, south}\}$
a_1: vote $a_1 a_2 > a_1 = 0$
a_2: vote $a_2 a_2 > a_1 = -3$

The solution to the conflict in Figure 5a is $a_2 > a_1$. The new conflict between a_1 and a_3 is solved with the following dialogue:

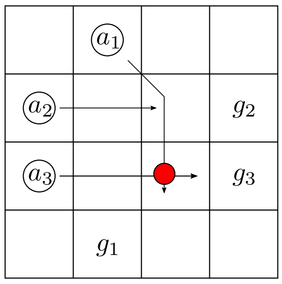
Opening stage
a_1: no earlier conflict
a_3: no earlier conflict
Proposal stage
a_1: propose $a_1 > a_3$
a_3: propose $a_3 > a_1$
Evaluation stage, starting with $a_1 > a_3$, new paths are shown in Figure 5d
a_1: new path
$p_{a_1,2} = \{\textit{south east, south}\}$
a_2: new path
$p_{a_3,2} = \{\textit{south east, east}\}$
a_2: vote$_{a_3}$ $a_1 > a_3 = -3$
a_1: vote$_{a_1}$ $a_1 > a_3 = -3$
Evaluation of $a_3 > a_1$, new paths are shown in Figure 5e
a_3: new path
$p_{a_3,1} = \{\textit{east, east}\}$
a_1: new path
$p_{a_1,3} = \{\textit{south west, south}\}$
a_1: vote $a_1 a_2 > a_1 = -3$
a_2: vote $a_2 a_2 > a_1 = -3$



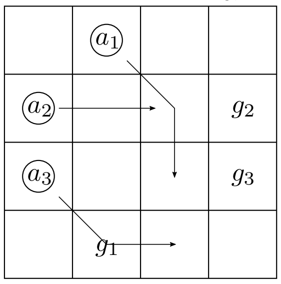
(a) Initial configuration showing $p_{a_1,1}$, $p_{a_2,1}$ and $p_{a_3,1}$.



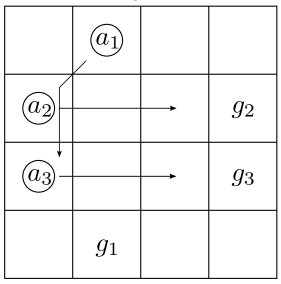
(b) Configuration after proposal $a_1 > a_2$. Shown are paths $p_{a_1,1}$, $p_{a_2,2}$, $p_{a_3,1}$.



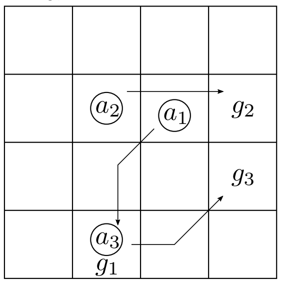
(c) Configuration after proposal $a_2 > a_1$. Shown are paths $p_{a_1,2}$, $p_{a_2,1}$, $p_{a_3,1}$.



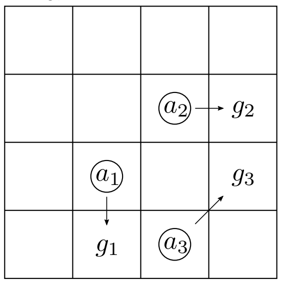
(d) Configuration after proposal $a_1 > a_3$. Shown are paths $p_{a_1,2}$, $p_{a_2,1}$, $p_{a_3,2}$.



(e) Configuration after proposal $a_3 > a_1$. Shown are paths $p_{a_1,3}$, $p_{a_2,1}$, $p_{a_3,1}$.



(f) Configuration after agents execute half of their plan ($p_{a_1,2}, p_{a_2,1}, p_{a_3,2}$).



(g) Configuration after agents have executed half of their plan for the second timestep.

Figure 5: Seven stages of resolving a conflict using WDPCA*. Agents are circles inscribed by a_i, their respective goals are g_i. Paths that are followed are indicated by the arrows. The circles indicate where agents have conflicting moves in their paths.

Table 3: Comparison of proposed cooperative pathfinding algorithms. The communication and online columns are the same as those in Table 1.

Algorithm	Communication	Online	Dialogues
PCA*	All	No	No
DPCA*	All	No	Yes
WDPCA*	Window	Yes	Yes

Both proposals have been evaluated equally so we break the tie in favour of a_1. The priority orderings for the first time step are $a_2 > a_1$ and $a_1 > a_3$. The agents now execute half of their plan, in this case that is one time step ($w/2$). After executing half of their plan the agents discard the priority orders $a_2 > a_1$ and $a_1 > a_3$ and they make new plans for the next 2 time steps. These new plans are shown in Figure 5f. There are no conflicts so no dialogues need to bee held. Agents execute the first step of their plans and make new plans. The situation at the next time step is shown in Figure 5g. There are again no conflicts. After all agents execute the first action in their plan they have all reached their goal and this instance of the cooperative pathfinding problem has been solved.

4.4 Summary

An overview of all proposed algorithms is given in Table 3, the categories shown the same as those in Table 1. Some of the columns from Table 1 are missing in Table 3 because these have the same values for all proposed algorithms. Each algorithm is decentralized since they do not rely on a central processor at any time. The algorithms are heavily influenced by the decoupled approach where agents first calculate optimal routes, then find a priority ordering and finally plan a route to their destination that adhere to the constraints that are imposed by the priority ordering. None of the methods is complete because they belong to categories that generally do not include complete algorithms. The meaning of the "communication" and "online" columns is the same as in Table 1. The "dialogues" column is new, it indicates whether agents can propose solutions, argue about them and share evaluations on the proposals.

5 Experimental results

The three proposed algorithms are tested and compared against the complete algorithm OD+ID and the fully decentralized algorithm DiMPP. To do this we compare them on a large set of problem instances. The exact size of the problem set depends on the experiment. Each instance in the set consists of a 16×16 8-connected grid. Each grid cell has a 20% chance of containing an impassable obstacle. Agents cannot enter these grid locations, but the obstacles do not block agents from moving along diagonals as explained in Section 2. Agents are placed randomly in the grid such that no two agents have the same starting position. Each agent is also given a randomly chosen destination location. These are also picked in such a way that no two agents have the same destination. Note that it is possible that the starting position of one agent might be the destination of another. All experiments were implemented in Python 3.6 and run on a single core of an AMD FX-8120 clocked at 3.1GHz.

Several experiments are done that compare the proposed algorithms to OD+ID and DiMPP. Before running these experiments the weights that determine the evaluation of proposals during dialogues need to be set. The empirically found weights are discussed in Subsection 5.1. The main experiments are carried out in Subsection 5.2. The main hypothesis for that section is that the proposed algorithms are able to solve more problem instances and are able to solve them faster than OD+ID. It is expected that OD+ID finds better quality solutions. The quality of a solution is primarily quantified as the cost of the solution. The number of loops that occur within an agent's path is used as a secondary measurement of the quality of a solution. We expect that the proposed methods perform on an equal or slightly worse level than DiMPP on all of these aspects. The number of agents in a problem instance may also influence the ability of the algorithms to find a solution within a reasonable time limit so we also measured this. Finally the effect of using a path cache on the time to find a solution is also recorded. This is not compared to OD+ID or DiMPP because these do not support similar techniques.

5.1 Evaluation weights

DPCA* and WDPCA* both need to evaluate proposals made by agents. The evaluation of a proposal is based on the effects that they have on an agent's plan and how many conflicts the agent will have to solve in the future. This is in turn used to cast a vote on the suitability of a proposal. To be able to evaluate a proposal properly there are several weights that need to be set so that each effect is taken into account sufficiently without having an exaggerated effect on the final evaluation. To set the weights simulated annealing [15] is used with an initial temperature of 100 which drops to 1 in unit steps. Each iteration of the simulated annealing process consists of 200 problem instances which contain between 2 and 50 agents. The number of agents in the problem instances forms a uniform distribution. For each drop in temperature a new unique set of 200 problem instances is generated to avoid overfitting to a certain set of problems. There is a time limit of 2000ms to solve each instance. Note that the time limit is arbitrary, it was picked to show results that can be expected in an application while not spending too much time on very complex instances that require a lot of computation.

The results of the simulated annealing process are shown in Table 4. The path length weight refers to the increase in path length that lower priority agents suffer when they have to take a new longer path. The weight is multiplied with the number of steps that the new path is longer than the path before solving a conflict. Some proposals have the side effect of creating or solving additional conflicts which is weighted by the conflicts solved weight. Each additional conflict increases the evaluation by this number while each solved conflict reduces the vote by this number. Each proposal is expected to reduce the number of conflicts by one, this is the conflict that is currently being discussed. The partial solved weight is used in cases where multiple agents take part in a dialogue and the proposal that is being evaluation only assigns a priority to some of the agents in the conflict but this doesn't solve the conflict for all agents involved.

Table 4: Evaluation weights as determined by simulated annealing. Empty fields mean that the weight is not used by that algorithm.

	Path length	Conflicts solved	Partial solution
DPCA*	4.744	5.291	
DPCA*+	0.312	5.570	2.677
WDPCA*-2	3.113	9.464	
WDPCA*-4	8.736	7.9143	
WDPCA*-8	9.352	22.874	

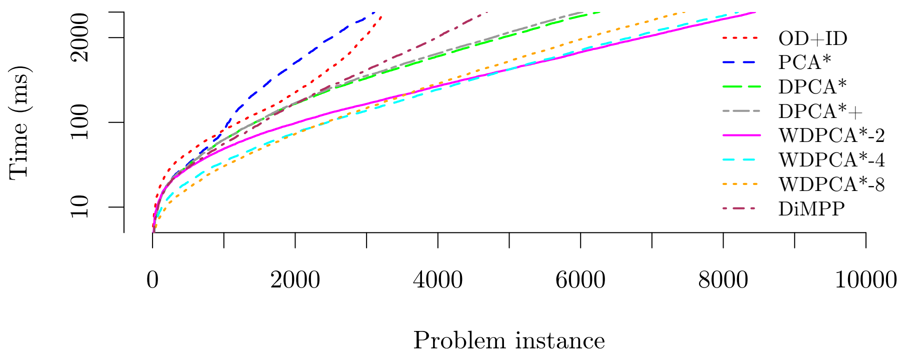


Figure 6: Comparison of performance of various algorithms on 10000 problem instances.

5.2 Experimental evaluation

To compare the algorithms they are all given a large set of cooperative pathfinding problems that have to be solved. Each instance is constructed following the same procedure as used to find the optimal evaluation weights. Each instance has between 2 and 40 agents, the number of agents still forms a uniform distribution. The time it takes to solve each instance in the set of problems is recorded. There is a time limit of 2000ms to calculate a solution for each instance. This is the same tile limit as that was used to find optimal evaluation weights.

When the instances are sorted in ascending order by the time it requires to solve them then they can be plotted in a *performance graph* as shown in Figure 6. The x-axis shows the index of the sorted instance while the y-axis shows the time it takes an algorithm to solve that instance. This sorting is done for each algorithm independently. Because of this it is not necessarily the case that the nth instance for one algorithm is the same as the nth instance for a different algorithm. The graph shows several different types of information. The first is a comparison of run times for the different algorithms. A lower line means that an algorithm was able to solve problem instances quicker. At the same time the graph also shows how many instances an algorithm can solve within the 2000 ms time limit. Instances that were not solved are not included in the graph, so how far a plot extends along the x-axis indicates how many problem instances were solved by an algorithm. A plot that extends further to the right indicates that an algorithm was able to solve more instances than a plot that does not extend as far.

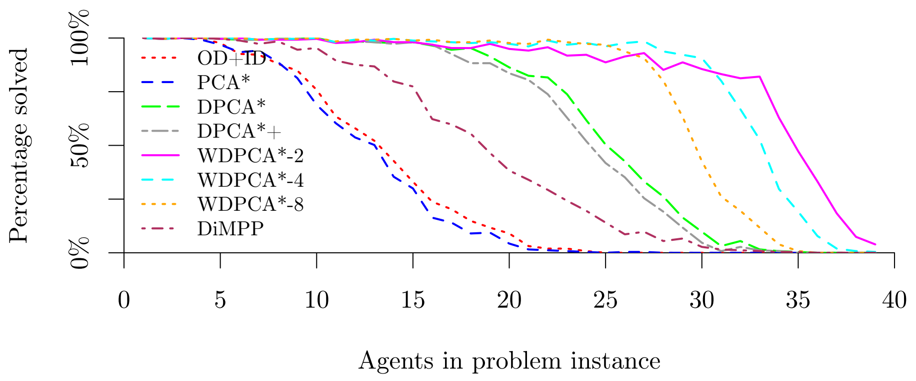


Figure 7: Proportion of instances that were solved by each algorithm.

The performance graph in Figure 6 shows that only the PCA* is generally slower than the complete algorithm OD+ID. When looking at individual instances we can see that 60% of the instances that were solved by both PCA* and OD+ID solved the latter was faster. All other methods are faster than OD+ID and are able to find a solution to more instances. The performance of DiMPP is similar to that of DPCA* and DPCA*+. Up to 2200 instances DiMPP is faster than the two proposed algorithms. After that point the algorithm is slower and it is not able to solve as many problem instances. The DPCA* and DPCA*+ versions of the algorithm have an almost equal performance. DPCA* is slightly faster on each individual instance by 34 ms as shown by a one-sided paired t-test ($t(5979) = -22.96, p = 3.95 \times 10^{-112}$). They are able to solve about the same number of instances, DPCA* solves 62.6% of instances while DPCA*+ is able to solve 60.3%. This indicates that the added complexity in DPCA*+ does not decrease the time required to solve a problem instance.

The size of the window also has a small influence on the performance. From Figure 6 we can see that the use of any window means that the algorithm is faster and is able to solve more problem instances. We can also see that the graphs for different window sizes are similar to each other. There seems to be a trade-off between the size of the window and the time to solve instances. A larger window is able to solve early instances faster, but after about 3500 instances it takes more time to solve problems than when a smaller window is used. An analysis of variance test shows that there is a difference in the time to reach a solution for different window sizes ($F(2, 24103) = 35.25, p = 5.15 \times 10^{-16}$). Individual t-tests show that $w = 2$ differs from both $w = 4$ and $w = 8$ in run time ($t(16613) = 2.41, p = 0.0161$ and $t(15540) = 3.69, p = 0.000228$) respectively. There is no difference between $w = 4$ and $w = 8$ ($t(15488) = 1.33, p = 0.183$). This confirms that a very limited window show different behaviour from larger windows. Especially the earlier instances in Figure 6 may have been of influence on this result. The gain in performance by using WDPCA* over DPCA* seems to be most important overall effect. The individual differences between various window sizes are important when considering other effects.

Figure 7 shows how many instances each algorithm can solve within the time limit for varying number of agents in an instance. It shows that there is indeed

Table 5: Solution quality of algorithms. Length is the sum of the lengths of the paths for a single problem instance.

	Instances solved	Length
OD+ID	32.6 %	73.56
PCA*	31.1 %	69.74
DPCA*	62.6 %	121.89
DPCA*+	60.3 %	117.58
WDPCA*-2	84.5 %	172.78
WDPCA*-4	82.1 %	164.23
WDPCA*-8	74.6 %	147.52
DiMPP	46.9 %	101.21

a little difference in performance of WDPCA* when the size of the window is varied. A smaller window means that instances with more agents are more likely to be solved. It also shows that there is a trade-off between a smaller window and the ability to find solutions. WDPCA* with a small window is not able to solve as many instances with a medium amount of agents as WDPCA* with a larger window. WDPCA*-2 is not able to solve all instances from 10 agents onwards while for WDPCA*-8 this happens from 22 agents, just before there is a cliff in the percentage of instances solved. The graph also shows a similar picture as Figure 6 where OD+ID and PCA* solved only few instances while DPCA* and its windowed variants are able to solve problems that include more agents. DiMPP is in between PCA* and DPCA* in terms of the number of agents it can find a solution for within the time limit. Figure 7 shows that DPCA* and DPCA*+ have the same overall performance. They both solve the same fraction of instances for any number of agents. This again suggests that the added complexity of DPCA*+ does not mean that it is able to solve more instances of the cooperative pathfinding problem.

As shown by Table 5 PCA* is the only version of the algorithm that solves fewer instances than OD+ID and DiMPP. All versions of DPCA* and WDPCA* are able to solve at least double the amount of instances as OD+ID. A smaller window means that the algorithm is able to solve more problem instances. This comes at a trade-off as a smaller window size also results in a larger solution cost. This can also be seen in Figure 8 where the sum of the path lengths is plotted as a function of the number of agents in a problem. It shows that WDPCA* finds longer paths than any other algorithm tested while OD+ID and DPCA* calculate shorter routes for the same number of agents in a problem. The graph for OD+ID shows heavy fluctuation between 20 and 26 agents. This may be because this algorithm was not able to solve all instances with this number of agents and is more sensitive to outliers because of this.

The length of the paths alone do not tell the full story of the quality of the solution. Agents may have a node N in its path multiple times. This can either be because an agent has a *wait* action on that node or it can be because there is a loop in an agent's path. In this case the agent visits other nodes between two visits of node N. To analyse the quality of the paths the number of loops in each problem instance was recorded. Only the loops that occur before the agent first visits its destination node were counted. After an agent reaches its goal node there is still the possibility of loops, however these loops mean that

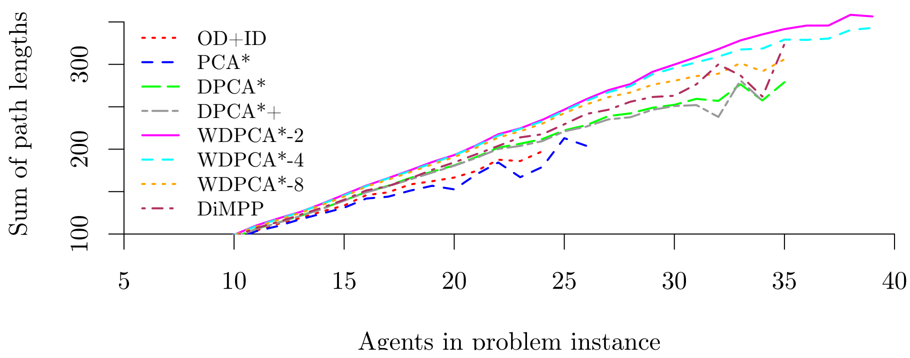


Figure 8: Mean sum of path length versus the number of agents in the problem instance. The results for 2 to 10 agents have been cut of because the path lengths for all algorithms are near identical for those amount of agents.

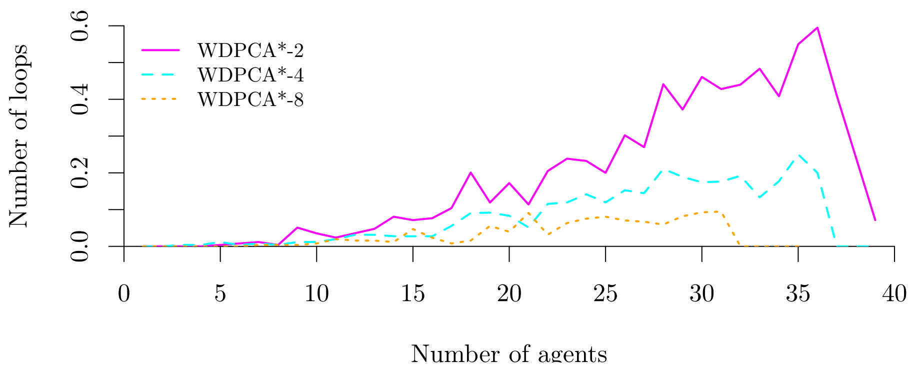


Figure 9: Number of loops in paths.

the agent has moved off its goal node to allow other agents to pass through. These loops are desired behaviour that occur when agents cooperate. We also do not count *wait* actions as loops, so only the number of unique times that an agent visits a non-goal node are counted.

The mean number of occurring loops for each number of agents is shown in Figure 9. Only the results for WDPCA* are shown, no loops were found in the solutions provided by OD+ID, DiMPP and other versions of the proposed algorithm. They plan a path from the starting position to the goal position, loops can be avoided because agents have full knowledge what their plan will be before executing it. We can see that there is an increase in the mean number of loops as the number of agents in a problem instance increases. How many loops occur depends on the size of the window that has been utilised. A larger window means that agents have a larger view of the world and are better able to find a solution where agents get closer to their goal without having to backtrack.

The average number of dialogues that have taken place to find a solution is shown in Figure 10. It shows that more dialogues are needed when the problem instance contains more agents. DPCA* and DPCA*+ are very similar again, so there is little difference in this aspect of the algorithms as well. Figure 10

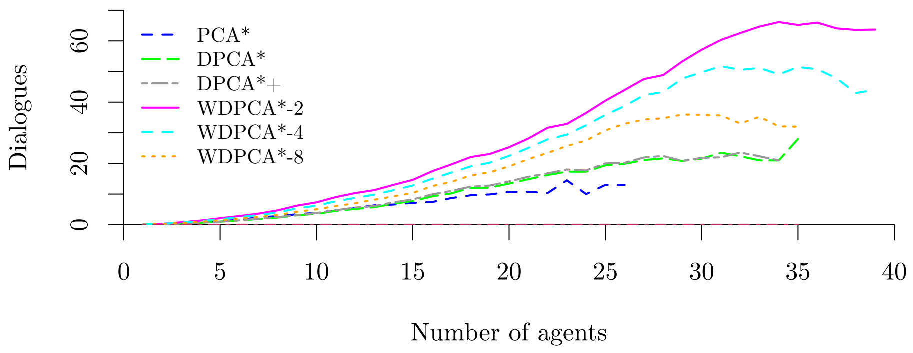


Figure 10: Mean number of dialogues that needed to be completed to find a solution for instances with n agents.

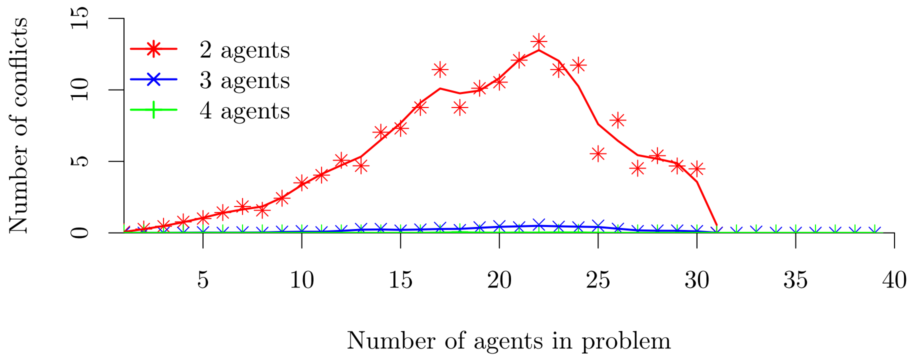


Figure 11: Comparison of occurence of different sizes of DPCA*+ dialogues.

also shows that more dialogues take place when a window is applied to conflict resolution, and with a decreasing size in window there are more dialogues that take place. One thing that is noticeable is that the number of dialogues required increases when the number of agents increases. At some point this trend trails off. Depending on the algorithm this is generally when there are between 25 and 30 agents in a problem instance. There is a peak in the number of dialogues between 28 and 35 agents.

The similarities in performance of DPCA* and DPCA*+ can be explained by Figure 11. It shows the number of dialogues that occur depending on the number of agents in a conflict. The number of conflicts is split by the number of agents that are part of the conflict. It shows that most conflicts occur between two agents and that most dialogues thus only have two agents participating. It is not a given that there are dialogues with more agents involved in any problem instance. The peak of the number of dialogues with three or more agents in a single has a value of 0.577. For dialogues with four agents this is even lower 0.0769 dialogues per instance at most. This means that in most problem instances there will only be dialogues in which two agents try to solve a conflict. Only rarely do dialogues with more than two agents occur. DPCA*+ often doesn't have to solve more complex dialogues, but it uses a more complex

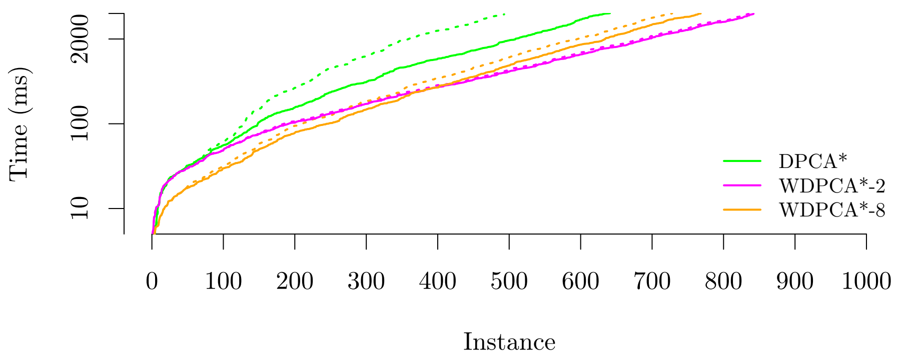


Figure 12: Comparison of performance of various algorithms with and without the use of a path cache. Dotted lines indicate that path cache wasn't used while solid lines indicate that the cache was used.

strategy to reach a successful conclusion for each dialogue and therefore it is slightly slower than DPCA*.

The agents make use of a path cache to reduce the amount of calculation required when they resolve conflicts. If the cache is in use then there should be a noticeable decrease in the time it takes to solve problems when compared to when agents don't cache paths. To measure the effect of caching on solving cooperative pathfinding problems a separate experiment was conducted. A new set of 1000 problem instances was generated. Each instance had between 2 and 40 agents in it. Each instance was presented to DPCA* and WDPCA* twice: once with the path cache disabled and once with the cache enabled. Similar to other experiments the algorithms had to solve each instance within 2000ms.

The performance graph of this experiment is shown in Figure 12. Only the effects of the cache on DPCA* and WDPCA* with $w = 2$ and $w = 8$ are shown to reduce visual clutter. The effects on DPCA*+ and WDPCA* with $w = 4$ were also tested. The solid lines in the plot represent the performance of an algorithm with caching enabled, while the dotted line shows the performance with the cache disabled. It shows that using a cache has a large effect on DPCA*, but not on WDPCA*-2. An analysis of variance shows that caching does have an effect on the time to solve an instance ($F(1, 326) = 22.51, p = 3.13 \times 10^{-6}$). So there is indeed a significant improvement in the performance of an algorithm when the cache is employed. There is also an interaction between the algorithm and whether the cache is enabled ($F(4, 326) = 4.21, p = 0.00245$). This confirms that the effectiveness of the cache depends on the algorithm that is used. Using no window or a large window benefit more from employing a cache than when a small window is used.

6 Discussion

This section will first discuss the experimental results on a more general level. Next the implications of our algorithm and its relation to the literature presented in Section 3 will be discussed. Finally we make some suggestions for future research.

6.1 Results

We have seen that only PCA* takes more time to solve problem instances and solves fewer instances than OD+ID and DiMPP. All other versions of the proposed algorithm can solve more instances and are generally faster than the two reference algorithms. This comes with the caveat that DPCA* and especially WDPCA* find solutions for which the cost is higher than those found with the algorithms from the literature. It is expected that the proposed algorithms are faster than OD+ID since the latter is a centralized algorithm while the proposed algorithms are all decentralized. Centralized methods are generally slower than decentralized algorithms because they consider the combined state spaces of all agents. Decentralized methods only use the state space of a single agent to create a plan for that agent. This results in a reduction in the complexity of the problem [1, 27].

Although centralized methods are slower than decentralized methods they find a solution in which each agent's path is optimal. They are also complete; given enough time they will find a solution to the problem if one exists. The experimental setup limits the amount of time that an algorithm can use to find a solution which results in complete methods like OD+ID not being able to find a solution. Instead the experiments are focused on finding a solution in as little time as possible. We consider being able to find a solution in a reasonable time to be more important than being able to find an optimal solution when given hours, days, or even weeks to calculate a solution. In a real-time applications like video games or groups of autonomous robots it is more important to find any solution in reasonable time than it is to find an optimal solution [2]. Because WDPCA* is fully decentralized it is able to find a solution in a reasonable amount of time while sacrificing the quality of the solution. This means that WDPCA* is more suitable for real-time applications because it is able to resolve a problem instance in a minimal amount of time. It is also able to find a solution to problems with a large amount of agents because of this speed.

WDPCA* is able to solve more problem instances than DPCA* (which does not use a window), see Table 5 and Figure 7. How many more instances can be solved depends on the size of the window. As a trade-off the amount of actions required for agents to reach their destination increases. This effect get stronger when the window size gets smaller (see Figure 8). When $w = 2$ most of the instances were solved, however the sum of the path lengths was also the highest. The required time of WDPCA*-2 being so low is not surprising. With a small window the algorithm is reduced to be reactive. There is barely any global planning any more because agents don't look far ahead when trying to find their way to the destination. Agents mainly solve conflicts that will happen during their next action or the time-step directly after that. Agents will also determine whether they have conflicts after each time-step. This shows that it is a valid strategy to create many small plans with low computational effort instead of a complete global plan. Cooperating in too small a window does have a negative effect on the quality of the solution.

In several figures there is a clear initial trend which trails off when the number of agents in the problem instance becomes larger. The number of dialogues per number of agents in Figure 10 is an example of this. There is an initial increasing trend which flattens out in the last last quarter of each plot. The plots for WDPCA* break this trend starting from between 25 agents and 33

Table 6: Comparison of several cooperative pathfinding algorithms [29, 30, 27, 5, 28, 35, 7]. DPCA* and DPCA*+ are encapsulated by the DPCA* row.

	Category	Complete	Priority	Comm.	Online	Dial.
OD+ID	Centralized	Yes	No	All	No	No
ICTS	Centralized	Yes	No	All	No	No
IADPP	Decoupled	No	Yes	All	No	No
WHCA*	Decoupled	No	Yes	Window	Yes	No
DMRCP	Decentralized	No	No	2 nodes	Yes	No
DiMPP	Decentralized	Yes	Yes	Ring	No	No
PCA*	Decentralized	No	Yes	All	No	No
DPCA*	Decentralized	No	Yes	All	No	Yes
WDPCA*	Decentralized	No	Yes	Window	Yes	Yes

agents per instance. The point where the growth in the number of dialogues required to solve a problem decreases coincide with the point in Figure 7 where each algorithm has a cliff in the fraction of instances that have been solved. This suggests that the effect of the number of dialogues required to find a solution is caused by the algorithm not being able to find a solution to complex problem instances within the 2000ms time limit. Complex problems are those that require many dialogues to find a priority scheme that allows all agents to find a path to their destination. This in turn suggests that WDPCA* successfully finding a solution to a problem instance depends on the number of conflicts in a problem, not the number of agents.

There is an interaction between which algorithm is used and whether previously found paths were stored in a cache and reused. From Figure 12 it becomes clear that WDPCA* with a small window also means that the effect is small. When w is small there are fewer conflicts so agents will not have to participate in dialogues very often. This means that they will not consult the cache as often and therefore there is less of a speed-boost. On top of that the paths are also shorter and easier to calculate. Retrieving a short path which consists of two or four actions from the cache is not much faster than calculating the path outright.

6.2 Implications

A complete comparison of the algorithms discussed and evaluated is given in Table 6, it combines the information found in Table 1 and Table 3. The category column indicates the approach the algorithm takes to solve a problem. The complete column indicates whether an algorithm is guaranteed to find a solution if one exists. The priority column indicates whether agents are assigned a priority ordering before calculating a solution. The communication column indicates which agents are allowed to communicate with each other. All means that there are no limits on communication, window means that agents can communicate with each other in a certain range, 2 nodes means that agents communicate in a 2 graph node radius, ring means that agents communicate in a chain which forms a ring. The online column indicates whether planning and execution is interleaved. The dialogues column indicates whether agents can influence the solution that is found, if the value is ‘no’ then the solutions found are abstract.

The online algorithms presented in Table 6 are faster than the state-of-the-art OD+ID [29, 35]. Two of these, WHCA* and WDPCA*, use a window to interleave planning and pathfinding. By limiting how far into the future agents cooperate we can speed up the time to find a solution. These results have also been found by research into Continual Planning [4]. This makes online algorithms ideal candidates to solve the cooperative pathfinding problem. Previous research and our results show that this comes at a trade-off: the paths that are found are often not optimal and the agents may display unintelligent behaviour like loops in their paths. The cause of the lower quality solution is that when w is small the algorithm becomes more reactive. Agents will move towards their goal and notice that they have a conflict for which the best solution is to backtrack. It may also occur that agents will move back to an earlier position in their path because they had to move out of the way of another agent. This kind of behaviour can be prevented by allowing agents to look further ahead so they can coordinate their actions earlier to prevent backtracking. This shows that there is a clear trade-off between finding a solution in a low amount of time and finding a low cost solution with few loops.

Conventional algorithms find an abstract solution for a pathfinding problem based on minimal cost. DPCA*, DPCA*+ and WDPCA* add transparency to the solution finding process by the dialogues in which agents can put forward arguments for or against partial priority orderings. Agents also evaluate and vote on each proposal based on several criteria. This gives agents some influence over which solution is picked for a problem instance. For an outside observer, the agents' arguments and evaluations provide an explanation why a group of agents have picked a particular solution. This can be used to explain to a user why a certain solution is more preferred than any other possible solution.

WDPCA* is based on the A* algorithm [13] but this can be changed to any pathfinding algorithm. Dialogues result in a priority ordering for agents, which in turn determines which agents should be considered moving obstacles by other agents. This is independent from which path planning algorithm is used. There are only constraints on where an agent can move to. As long as an algorithm is able to handle moving obstacles or can be modified to handle moving obstacles then it can be used instead of A* in WDPCA*. This means that our algorithm can be adapted to work with other discrete space algorithms, or even continuous space algorithms like RRT* [17, 18, 6].

The proposals of priority orderings are evaluated by the agents to find the best priority ordering. To do this they weigh different effects of the proposal. Currently these weights are static. The weights that were found using simulated annealing are a one-size-fits-all solution. The optimal weights that are found by simulated annealing are dependent on the lower and upper bound of the number of agents that can be in a problem instance. This suggests that the optimal value of the weights depends on the number of agents in the problem. Future work may look into setting weights as a function of the number of agents in the problem. WDPCA* could use a set of weights based on the number of agents in the current window. Instead of basing the weights on the number of agents the number of conflicts that an agent has can determine their value. This may make the agents respond appropriately to the complexity of a problem instance. Instead of using a one-size-fits-all solution the agents will adapt their heuristics to the problem instance.

The model of deliberation dialogues as proposed by McBurney, Hitchcock and Parsons (MHP model) [19] is quite complex. Walton et al. [34] have expanded on this so that the human deliberation dialogues can be modelled more accurately while addressing some of the shortcomings of the MHP model. The resulting model is a good explanation of how human deliberation dialogues work. The complexity of the model is not a good fit for a multi-agent coordination problem like the one presented in this thesis. Instead the simpler TeamLog model [10] is more appropriate for this setting. The MHP model and the TeamLog model both consists of an opening part, a closing part, and an argumentation part where all deliberation takes place. The main difference is in the number of stages in the argumentation part: the MHP model consists of four stages while TeamLog consists of just two stages. These two stages are sufficient to be able to solve the conflicts that occur in the cooperative pathfinding problem. For WDPCA* we simplified the model even more by allowing agents to only make proposals and evaluate them. The discussion that might occur in a more complex model is contained within the evaluation stage. This simplicity means that WDPCA* is efficient at finding a solution, but it sacrifices richness in the dialogue.

Richness in dialogues also depends on the domain. Our problem formulation is very abstract and generic. The only argument that agents can make against a priority proposal is that there is no valid path to their destination. Applications like traffic management can impose additional constraints on the problem. In air and rail traffic management agents may have to keep to a schedule. Agents can make arguments about keeping to their schedule and weigh the effects of possible delays. In multi robot systems the agents may need to recharge and they should be able to make arguments about why they need to have a high priority so that they can reach a charging station without too much delay. Applications may have constraints like these which do not fit well in an evaluation function, but they are well expressed in a logic. This also gives an even greater explanatory power than that WDPCA* has with its evaluation function and its limited argumentative capability.

The argumentative methods DeLP-POP and DeLP-MAPOP [22, 12] are general approaches to multi-agent coordination that allow agents to discuss individual actions in a deliberation dialogue. This makes the method of finding a solution analogous to OD+ID. This is in contrast to our approach in which agents deliberate on the level of priorities and paths (sequences of actions). The benefit of DeLP-MAPOP is that it is distributed, complete and optimal. It can find an optimal solution without relying on a central processor. However the deliberation process is quite complex and agents will often have to keep track of the same information. It is thus not the case that agents will only know about what influences their paths but also what influences the paths of others, something which is not the case in WDPCA*.

Cooperative pathfinding is a specific instance of a coordination problem. It is possible to generalise the findings here to other resource sharing problems. Instead of agents making moves in a grid world the agents would claim the use of a resource for some amount of time. Two agents have conflicting claims when they try to claim the same resource at the same or overlapping times. They can resolve this conflict in claims by starting a deliberation dialogue and make arguments about why an agent should be allowed to access the resource before the other. They could also make proposals about how to resolve the conflict in claims and agents should be able to argue for or against its adaptation. A voting system similar to the priority scheme evaluation used by WDPCA* could also be employed.

6.3 Future research

The family of algorithms we have presented here show some good results. Some improvements to our algorithms can be made. It was suggested that the optimal voting weights may depend on the number of agents present in the problem. From casual observation it seems that there is no single set of weights that gives optimal performance. Exploring whether speed performance and path quality increase when the voting weights depend on the number of agents may be worthwhile. For WDPCA* there may even be a difference between weights dependent on the number of agents in the problem or the number of agents in a window.

Some improvements may be made in which dialogues agents can participate in and which arguments they can put forward. When agent a_i and a_j have coordinated in the past they may want to form an alliance against agent a_k if either of them would have a conflict with a_k. By doing so agents that have locally well coordinated plans could force other agents to adjust to their plans. This would avoid the need to readjust plans of multiple agents if one of them would have to adjust. Contrary a_k may also want to weigh in on a dialogue if certain priority ordering proposals would have a negative impact on its plan. In Figure 4 agent a_3 may want to weigh in on initial the conflict resolution dialogue between a_1 and a_2. This could allow it to determine which proposal would be accepted during that dialogue.

We have looked at path level argumentation but it is also possible for the agents to argue about individual actions similar to DeLP-MAPOP. This would in itself allow for richer dialogues as agents can put arguments forward about why certain actions in its plan are important. It may also allow agents to negotiate about their actions. Currently there is not much room for negotiation but this may change when agents can switch having priority and yielding based on their circumstances. It would also allow the algorithm to be complete. Certain situations require agents to alternate yielding and having priority, something which is almost impossible when a hierarchy is imposed on the agents. Dialogues about individual actions are more likely to result in a complete algorithm.

Finally a better formalisation would allow some properties of our algorithms to be proven. This includes the time and space complexity of the algorithms and whether they are guaranteed to terminate. Currently it is possible to provide an informal proof of termination at best. A more formal definition would also allow other improvements, such as completeness, to be made more easily.

7 Conclusion

In this thesis we have combined ideas from partial global planning, continual planning and computational argumentation and applied them to cooperative pathfinding. A new novel algorithm called WDPCA* was developed starting from the decoupled model for cooperative pathfinding. Pairs of agents solve conflicts in their paths by finding determining a priority order. One agent will have to give priority to the other and consider it to be a moving obstacle. Partial global planning is used to allow agents to incrementally build their private priority order. Doing so means that the agents will coordinate their paths so that there are no conflicts between them. Resolving conflicts is done in a deliberation dialogue. During such a dialogue agents will discuss possible priority orders that lead to a solution to the conflict. If there is more than one possibility they will vote on which priority scheme they prefer. The resulting priority scheme is adapted by both agents that participate in a dialogue.

Generally agents will resolve all their conflicts before they start to execute their plan. This has several limitations: all computation is done before execution, and finding a solution may require agents to alternate giving and having priority. The most important limitation is that agents will stop cooperating once they reach their goal. This may mean that they block the paths of early agents and do not cooperate with agents that need to pass them. To address these limitation a window is applied to the search. This limits the range in which conflict resolution takes place. It also ensures that agents will continue to cooperate after they have reached their destination by forcing them to keep planning even though they have already reached their goal.

We have shown that WDPCA* is able to solve problem instances faster than the well established algorithm OD+ID and the recent algorithm DiMPP. It is also able to solve problem instances with a large number of agents where both of the reference algorithms were not able to solve these instances. As a trade-off the paths found by WDPCA* are often longer and may contain loops where agents visit the same grid cell multiple times. Employing a large window results in fewer of these low quality paths while it does not sacrifice much of the speed gain that a small window has.

The agents solve conflicts by through a deliberation dialogue. This makes it possible to retrieve arguments why agents have picked a certain priority ordering. These arguments explain why the solution to the cooperative pathfinding problem was picked. Because WDPCA* uses arguments it is possible to integrate domain specific knowledge into the algorithm. The method we proposed here is less abstract than conventional cooperative pathfinding algorithms which only calculate a solution without giving arguments why that solution is the most appropriate.

References

[1] Maren Bennewitz, Wolfram Burgard, and Sebastian Thrun. Finding and Optimizing Solvable Priority Schemes for Decoupled Path Planning Techniques for Teams of Mobile Robots. *Robotics and Autonomous Systems*, 41(2–3):89–99, 2002.

[2] Adi Botea, Bruno Bouzy, Michael Buro, Christian Bauckhage, and Dana Nau. Pathfinding in Games. In Simon M. Lucas, Michael Mateas, Mike Preuss, Pieter Spronck, and Julian Togelius, editors, *Artificial and Computational Intelligence in Games*, volume 6 of *Dagstuhl Follow-Ups*, pages 21–31. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013.

[3] Michael Brenner and Ivana Kruijff-Korbayová. A Continual Multiagent Planning Approach to Situated Dialogue. *Proceedings of the 12th Workshop on Semantics and Pragmatics of Dialogue (LONDIAL)*, pages 61–68, 2008.

[4] Michael Brenner and Bernhard Nebel. Continual planning and acting in dynamic multiagent environments. *Autonomous Agents and Multi-Agent Systems*, 19(3):297–331, 2009.

[5] Michal Čáp, Peter Novak, Jiří Vokřínek, and Michal Pěchouček. Asynchronous Decentralized Algorithm for Space-Time Cooperative Pathfinding. In Mehul Bhatt, Hans W. Guesgen, and Ernest Davis, editors, *Spatio-Temporal Dynamics (STeDy 2012), Workshop Proceedings of the European Conference on Artificial Intelligence (ECAI 2012), Montpellier, France*, pages 18–25. ECAI, August 2012.

[6] Michal Čáp, Peter Novák, JiYí Vokrínek, and Michal Pěchouček. Multi-agent RRT: Sampling-based Cooperative Pathfinding. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '13, pages 1263–1264, Richland, SC, USA, 2013. International Foundation for Autonomous Agents and Multiagent Systems.

[7] Satyendra Singh Chouhan and Rajdeep Niyogi. DiMPP: a complete distributed algorithm for multi-agent path planning. *Journal of Experimental & Theoretical Artificial Intelligence*, 29(6):1129–1148, 2017.

[8] Keith S Decker and Victor R Lesser. Generalizing the Partial Global Planning Algorithm. *International Journal of Intelligent and Cooperative Information Systems*, 1(02):319–346, 1992.

[9] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–357, 1995.

[10] Barbara Dunin-Kęplicz, Alina Strachocka, and Rineke Verbrugge. Deliberation Dialogues During Multi-Agent Planning. In *Proceedings of the 19th International Conference on Foundations of Intelligent Systems*, ISMIS 2011, pages 170–181, Berlin, Heidelberg, 2011. Springer-Verlag.

[11] Edmund Durfee and Victor R. Lesser. Partial Global Planning: A Coordination Framework for Distributed Hypothesis Formation. *IEEE Transactions on Systems, Man, and Cybernetics*, 21:1167–1183, 1991.

[12] Sergio Pajares Ferrando and Eva Onaidia. Defeasible Argumentation for Multi-Agent Planning in Ambient Intelligence Applications. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems – Volume 1*, pages 509–516. 2012.

[13] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, July 1968.

[14] John E. Hopcroft, Jacob T. Schwartz, and Micha 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, 1984.

[15] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[16] Jean-Claude Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Norwell, MA, 1991.

[17] Steven M. LaValle. Rapidly-Exploring Random Trees: A New Tool for Path Planning. 1998.

[18] Steven M LaValle and James J Kuffner Jr. Randomized Kinodynamic Planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

[19] Peter McBurney, David Hitchcock, and Simon Parsons. The Eightfold Way of Deliberation Dialogue. *International Journal of Intelligent Systems*, 22:95–132, 2007.

[20] Peter McBurney and Simon Parsons. *Dialogue Games for Agent Argumentation*, pages 261–280. Springer US, Boston, MA, 2009.

[21] Sanjay Modgil, Francesca Toni, Floris Bex, Ivan Bratko, Carlos I Chesñevar, Wolfgang Dvořák, Marcelo A Falappa, Xiuyi Fan, Sarah Alice Gaggl, Alejandro J García, María P González, Thomas F Gordon, Leite João, Martin Možina, Chris Reed, Guillermo R Simari, Stefan Szeider, Paolo Torroni, and Stefan Woltran. The Added Value of Argumentation. In Sascha Ossowski, editor, *Agreement technologies*, volume 8 of *Law, Governance, and Technology Series*, chapter 21, pages 357–403. Springer, Dordrecht, 2013.

[22] Pere Pardo, Sergio Pajares, Eva Onaidia, Lluís Godo, and Pilar Dellunde. Multiagent Argumentation for Cooperative Planning in DeLP-POP. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 3*, pages 971–978. 2011.

[23] John L Pollock. *Cognitive Carpentry: A Blueprint for How to Build a Person.* Mit Press, Cambridge, MA, USA, 1995.

[24] Henry Prakken. Models of Persuasion Dialogue. In Guillermo Simari and Iyad Rahwan, editor, *Argumentation in Artificial Intelligence*, pages 281–300. Springer US, Boston, MA, 2009.

[25] Prakken, Henry. Formal Systems for Persuasion Dialogue. *The Knowledge Engineering Review*, 21(2):163–188, june 2006.

[26] Iyad Rahwan and Guillermo Simari, editors. *Argumentation in Artificial Intelligence.* Springer, Dordrecht, 2009.

[27] Guni Sharon, Roni Stern, Meir Goldenberg, and Ariel Felner. The Increasing Cost Tree Search for Optimal Multi-Agent Pathfinding. *Artificial Intelligence*, 195:470–495, 2013.

[28] David Silver. Cooperative Pathfinding. In *Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'05, pages 117–122. AAAI Press, 2010.

[29] Trevor Standley. Finding Optimal Solutions to Cooperative Pathfinding Problems. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI, pages 173–178. AAAI Press, 2010.

[30] Trevor Standley and Richard Korf. Complete Algorithms for Cooperative Pathfinding Problems. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, volume 1 of *IJCAI*, pages 668–673. AAAI Press, 2011.

[31] Jur van den Berg, Ming Guy, Stephen J.and Lin, and Dinesh Monocha. *Reciprocal n-Body Collision Avoidance*, pages 3–19. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[32] Frans H. van Eemeren, Bart Garssen, Erik C. W. Krabbe, A. Francisca Snoeck Henkemans, Bart Verheij, and Jean H. M. Wagemans. *Handbook of Argumentation Theory.* Springer, Dordrecht, 2014.

[33] Douglas Walton and Erik CW Krabbe. *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning.* SUNY press, 1995.

[34] Douglas Walton, Alice Toniolo, and Timothy J Norman. Missing Phases of Deliberation Dialogue for Real Applications. In *Proceedings of the 11th International Workshop on Argumentation in Multi-Agent Systems*, pages 1–20, 2014.

[35] Changyun Wei, Koen V Hindriks, and Catholijn M Jonker. Altruistic Coordination for Multi-Robot Cooperative Pathfinding. *Applied Intelligence*, 44(2):269–281, 2016.

[36] Michael Wooldridge. *An Introduction to MultiAgent Systems.* John Wiley & Sons Ltd, 2nd edition, 2009.