Multi-Robot Cooperative Pathfinding: A Decentralized Approach

Changyun Wei, Koen V. Hindriks, and Catholijn M. Jonker

Interactive Intelligence Group, EEMCS, Delft University of Technology,
Mekelweg 4, 2628 CD, Delft, The Netherlands
{c.wei,k.v.hindriks,c.m.jonker}@tudelft.nl

Abstract. When robots perform teamwork in a shared workspace, they might be confronted with the risk of blocking each other's ways, which will result in conflicts or interference among the robots. How to plan collision-free paths for all the robots is the major challenge issue in the multi-robot cooperative pathfinding problem, in which each robot has to navigate from its starting location to the destination while keeping avoiding stationary obstacles as well as its teammates. In this paper, we present a novel fully decentralized approach to this problem. Our approach allows the robots to make real-time responses to the dynamic environment and can resolve a set of benchmark deadlock situations subject to complex spatial constraints in the robots' workspace. When confronted with conflicting situations, robots can employ waiting, dodging, retreating and turning-head strategies to make local adjustments. In addition, experimental results show that our proposed approach provides an efficient and competitive solution to this problem.

Key words: Cooperative pathfinding, coordination, collision avoidance.

1 Introduction

Autonomous robot teams are now expected to perform complicated tasks consisting of multiple subtasks that need to be completed concurrently or in sequence [1]. In order to accomplish a specific subtask, a robot first needs to navigate to the right location where the subtask can be performed. When multiple robots engage in such teamwork in a shared workspace, there is an inherent risk that the robots may frequently block each other's ways. As a result, the use of multiple robots may result in conflicts or interference, which will decrease overall team performance. In particular, deadlock situations have to be taken into consideration in highly congested settings for distributed or decentralized robots to plan their individual paths. This work is motivated by many practical applications such as unmanned underwater/ground vehicles, autonomous forklifts in warehouses, deep-sea mining robots, etc.

Interference in the Multi-Robot Cooperative Pathfinding (MRCP) (or called Multi-Robot Path Planning) problem stems from conflicting paths, along which multiple robots may intend to occupy the same places at the same time. The issue

of how to plan collision-free paths for multiple robots has been extensively studied in [2–7], where the robots are supposed to navigate to distinct destinations from their starting locations. Finding the optimal solution to such a problem, however, is NP-hard and intractable [3]. *Centralized* solutions are usually based on global search and therefore could guarantee completeness, but they do not scale well with large robot teams [4] and cannot be solved in real-time [8]. In many practical applications, robots are expected to be fully autonomous and can make their own decisions, rather than being managed by a central controller [9]. *Decoupled* (or called *distributed*) solutions are fast enough for real-time applications, but they usually cannot guarantee completeness [6], and the robots may easily get stuck in common deadlock situations. Recent decoupled advances [10, 2, 11] have considered highly congested scenarios. The essential feature of decoupled approaches is that the robots are connected by a common database, e.g., the reservation table in [10] and the conflict avoidance table in [2]. Comparatively, we are interested in fully *decentralized* solutions, where the robots explicitly communicate with one another to keep track of each other's states and intentions, instead of accessing the common database to be aware of the progress of their teamwork in decoupled solutions.

The advantage of decentralized solutions in multi-robot teams is that, as the robots do not use any shared database, they can be fully autonomous and each robot can be a stand-alone system. On the other hand, such a solution may encounter the problem of overload communication in large-scale robot teams. To deal with this problem, a robot in our approach only needs to communicate with the ones locating within its coordinated network. For a team of k robots, the communication load will be reduced to $12k$ times in each time step, rather than $k(k-1)$ times in traditional decentralized systems. We analyze various deadlock situations in a robot's coordinated network, and propose a pairwise coordination algorithm to identify which situation the robot is confronted with and which strategy it should adopt. Specifically, a robot can employ *waiting*, *dodging*, *retreating*, and *turning-head* strategies to make local adjustments to avoid collisions, following the estimated shortest path to its destination.

We begin our work by discussing the state-of-the-art approaches in Section 2. We analyze the models of our decentralized approach in Section 3, and then discuss our proposed algorithms in Section 4. The simulated experiments and results are discussed in Section 5. Finally, Section 6 concludes the paper.

2 Related Work

MRCP problems have also been studied in the context of exploration (e.g., in [12, 13]). The robots in [12] do not have long-term destinations to move; instead, they need to continually choose unexplored neighboring vertices to cover. A decision-theoretic approach is presented in [13], where the robots have long-term destinations but do not consider any congested configurations. Our work focuses on general MRCP problems, in which the robots need to navigate to

distinct long-term destinations from their starting locations in a highly congested environment.

Earlier work in [14, 15, 5] presented their decoupled solutions based on prioritized planning, where the robots need to plan respective paths in order of their priorities. The robot with the highest priority first plans its path without considering the other robots' locations and plans, and then this robot is treated as a moving obstacle so that subsequent robots have to avoid it when planning their paths later. In this approach, each robot is expected to find a path without colliding with the paths of higher priority robots, but the overall path planning algorithm will fail if there is a robot unable to find a collision-free path for itself. Particularly, such an approach does not respond well in highly congested scenarios such as intersections. The work in [16] introduced an approach to solving the two-way intersection traffic problem, which will not be considered as a congested setting in our work. Traffic problems usually have two-way roads and thus can be solved by introducing traffic lights, whereas one-way intersections cannot be simply solved using the traffic light theory.

Recent decoupled advances considering highly congested environments include FAR [17], WHCA* [10], MAPP [11], and ID [2]. FAR and WHCA* are fast and can scale well with large robot teams, but they are still incomplete and offer no guarantee with regard to the running time and the solution length. MAPP has an assumption that for every three consecutive vertices in a robot's path to its destination, an alternative path that does not go through the middle vertex should exist, which apparently does not suit a narrow corridor scenario. Only ID claims that it provides a complete and efficient algorithm by breaking a large MRCP problem into smaller problems that can be solved independently, so we only compare our approach with ID in the experimental study.

Our work takes the advantage of *push* and *swap* ideas proposed in [6], which presents a centralized approach that allows one robot to push the other one away from its path, or makes two robots switch their positions at a vertex with degree ≥ 3. However, this approach only allows one robot (or paired robots) to move in each time step. In contrast, we propose a novel decentralized approach that allows a robot to actively make local adjustments to cope with conflicting situations. In order to do so, the robot can employ *waiting*, *dodging*, *retreating*, and *turning-head* strategies in its own decision making process. This work focuses on general MRCP problems that are analyzed using graph-based models, so the motion planning problems with regards to low-level control parameters, such as acceleration, deceleration, turning angle, etc., are beyond the scope of this paper.

3 Models of Decentralized Cooperative Pathfinding

3.1 Problem Formulation

A shared workspace can be divided into a set of n discrete cells $\mathcal{V}$ with k robots $\mathcal{R}$, $k < n$, and we use an undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ to represent it. The edges $\mathcal{E}$ indicate whether two vertices are connected or not in $\mathcal{G}$. In order to model

conflicts among the robots, we define a spatial constraint: robots must be present at distinct vertices at any time t:

$$\forall i, j \in [1, k], i \neq j : \mathcal{A}_t[i] \neq \mathcal{A}_t[j], \tag{1}$$

where $A_t[i] \in \mathcal{V}$ indicates the vertex in which the i-th robot locates at time t. The sets of starting locations and destinations are denoted as $\mathcal{S} \subset \mathcal{V}$ and $\mathcal{T} \subset \mathcal{V}$, respectively. At any time t, each robot can have a next-step plan $\mathcal{P}_t$, either staying at its current vertex $\mathcal{A}_t$ or moving to one of its neighboring vertices $\mathcal{A}_{t+1}$:

$$\forall i \in [1, k], \mathcal{P}_t[i] \Rightarrow \begin{cases} \mathcal{A}_t[i] = \mathcal{A}_{t+1}[i] & \text{(stay at its current vertex,)} \\ (\mathcal{A}_t[i], \mathcal{A}_{t+1}[i]) \in \mathcal{E} & \text{(move to a neighbouring vertex.)} \end{cases} \tag{2}$$

Given the starting and destination vertices, when beginning to carry out the task, a robot (e.g., the i-th robot) needs to plan an individual path $\mathit{\Pi}_0[i] = \{\mathcal{S}[i], \ldots, \mathcal{T}[i]\}$ from its starting location $\mathcal{S}[i]$ to its destination $\mathcal{T}[i]$. At any time t, it may adjust its path to $\mathit{\Pi}_t[i] = \{\mathcal{A}_t[i], \mathcal{A}_{t+1}[i], \ldots, \mathcal{T}[i]\}$. Suppose the robot reaches its destination at time e, then we can know $\mathit{\Pi}_e[i] = \{\mathcal{T}[i]\}$. Therefore, the robot is supposed to minimize e while keeping avoiding collisions with stationary obstacles as well as the other robots.

3.2 Heuristic Estimated Shortest Paths

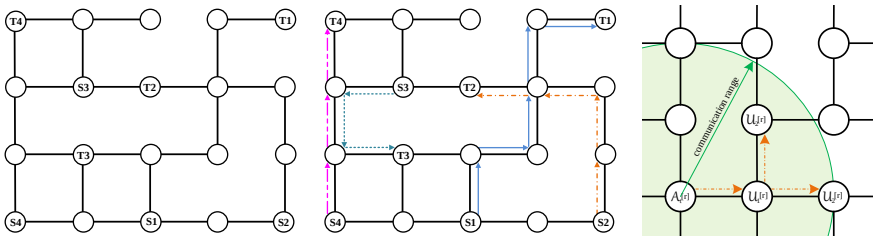
Fig. 1(a) shows an example of the initial configuration of the environment, where four robots need to go to their destinations. When planning their respective paths to the destinations, if each robot can disregard the presence of the others, they can easily find the shortest paths, as shown in Fig. 1(b). Such a path only considering stationary obstacles provides a *heuristic optimal estimate* with the shortest travel distance that can be calculated by performing a graph search, for example, using Dijkstra's algorithm in our work. Although such estimated paths cannot guarantee that a robot will successfully arrive at its destination without collisions with the other robots, it indeed provides an idealized estimate. We use $\mathcal{H}_t[i]$ to denote the heuristic estimated shortest path of the i-th robot at time t:

$$\mathcal{H}_t[i] = \{\mathcal{U}_1[i], \mathcal{U}_2[i], \ldots, \mathcal{T}[i]\}, \tag{3}$$

where $\mathcal{U}_1[i]$ and $\mathcal{U}_2[i]$ are the first and the second successor vertices in $\mathcal{H}_t[i]$, and thus $(\mathcal{A}_t[i], \mathcal{U}_1[i]) \in \mathcal{E}$, $(\mathcal{U}_1[i], \mathcal{U}_2[i]) \in \mathcal{E}$. When the robot's next step is its destination, then $\mathcal{H}_t[i] = \{\mathcal{T}[i]\}$, $\mathcal{U}_1[i] = \mathcal{T}[i]$ and $\mathcal{U}_2[i] = 0$.

3.3 Coordinated Network

Suppose $\mathcal{G}$ is divided uniformly, a robot takes unit length l to move to its neighboring vertices. For any robot $r \in \mathcal{R}$ at time t, it has the first and second successor vertices, $\mathcal{U}_1[r]$ and $\mathcal{U}_2[r]$, along its estimated shortest path (see Equation 3). At the moment, $\mathcal{U}_1[r]$ might be occupied by one of its teammates, or be



(a) (b) (c)

Fig. 1. Heuristic estimated shortest paths & coordinated network.

free but its teammate(s) is going to occupy it. Thus, we can know that if the next-step plan of robot r is conflicted with the one of its teammates s, s must be locating within the distance of $2l$, as shown in Fig. 1(c). Then each robot can construct a coordinated network with the communication range of $2l$, and it only needs to communicate and coordinate with the ones within its coordinated network. Normally, k decentralized robots may need to communicate $k(k-1)$ times in each time step, whereas in the coordinated network, the communication can be reduced to no more than $12k$ times.

4 Decentralized Multi-Robot Coordination

4.1 Main Decision Making Process

As robots make their own decisions in decentralized teams, we will discuss our proposed solution from a single robot's point of view. Algorithm 1 shows the main decision making process as to how robot r makes its next-step plans.

Algorithm 1 Main decision process for a single robot to plan its next step.

1: Given robot r and $\mathcal{T}[r]$ at time t, $\mathcal{A}_t[r] \leftarrow \mathcal{S}[r]$, $\Pi_t[r] \leftarrow \{\mathcal{S}[r]\}$.
2: **while** $\Pi_t[r] \neq \{\mathcal{T}[r]\}$ **do** ▷ not yet at destination.
3: **if** TURNING-HEAD $\neq 0$ **then**
4: $\mathcal{U}_1[r] \leftarrow$ TURNING-HEAD, $\mathcal{U}_2[r] \leftarrow 0$ ▷ turn head to a new robot.
5: **else**
6: Estimate shortest path $\mathcal{H}_t[r] = \{\mathcal{U}_1[r], \mathcal{U}_2[r], \ldots\}$, and ▷ see Equation 3.
7: Send $\mathcal{H}_t[r]$ ▷ broadcast its estimated path.
8: **end if**
9: **if** $\exists s \in \mathcal{R}$ such that $\mathcal{U}_1[r] == \mathcal{A}_t[s]$ **then** COORDINATION ▷ see Algorithm 2
10: **else if** $\exists s \in \mathcal{R}$ such that $\mathcal{U}_1[r] == \mathcal{P}_t[s]$ **then** ▷ s is entering the vertex.
11: $\mathcal{P}_t[r] \leftarrow \mathcal{A}_t[r]$ ▷ wait at current vertex.
12: **else** $\mathcal{P}_t[r] \leftarrow \mathcal{U}_1[r]$ ▷ move to the free successor vertex.
13: **end if**
14: Execute $\mathcal{P}_t[r]$ and Send $\mathcal{P}_t[r]$, $t \leftarrow t+1$
15: **end while**

Robot r will continuously make its next-step plans until it arrives at its destination (line 2). It either can choose the successor vertices along the estimated shortest path (see line 6), or needs to turn its head to a new coordinated robot

(line 4) if it took a *turning-head* strategy in the last round of the decision process (see Algorithm 2). If the next successor vertex of robot r is occupied by another robot s, then r needs to coordinate with s (line 9). Sometimes the successor vertex might not be occupied, but one of its teammates is moving to it now (line 10). In this case, the robot needs to wait at its current location for the next round of decision making. Otherwise, it can make the plan to move to the next free successor vertex (line 12). It happens that several robots may make decisions to go to the same vertex synchronously. In our approach, while executing a plan to move to a vertex, if a robot finds that there is the other one moving to the same vertex as well, the robot will immediately stop moving, give up its original plan and inform the other robot about its decisions.

4.2 Pairwise Coordination in Deadlock Situations

When the next successor vertex of a robot is occupied by one of its teammates, pairwise coordination between them is needed to make further progress. The robots may form various environmental configurations, and we list all the instances that may lead to a set of benchmark deadlock situations in Fig. 2.

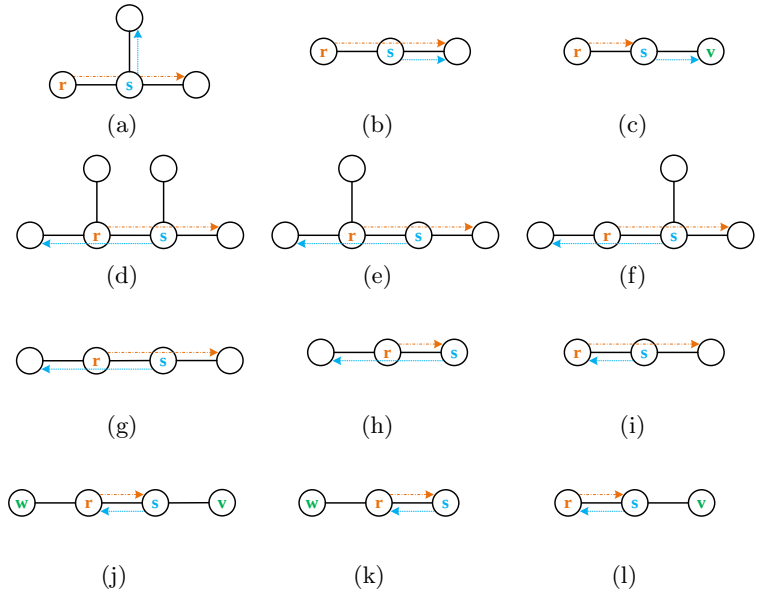


Fig. 2. Various environmental configurations in the pairwise coordination.

Trail following Fig. 2(a) and 2(b) depict that r is following the trail of s, $(\mathcal{U}_1[r] == \mathcal{A}_t[s]) \bigcap (\mathcal{A}_t[r] \neq \mathcal{U}_1[s])$. After the next successor vertex, r may diverge from s (see Fig. 2(a)), $\mathcal{U}_2[r] \neq \mathcal{U}_1[s]$, or still follow the trail of s (see Fig. 2(b)), $\mathcal{U}_2[r] = \mathcal{U}_1[s]$. In both instances, r will believe that s is trying to move away from the occupied vertex, so it can *wait* for s to succeed. Sometimes they may come to a situation, as shown in Fig. 2(c), where s cannot move away right now because

Algorithm 2 Pairwise coordination for making local adjustments.

Given robot r at time t, its current location $\mathcal{A}_t[r]$, and its first and second successor vertices $\mathcal{U}_1[r]$ and $\mathcal{U}_2[r]$, $\exists s \in \mathcal{R}$ such that $\mathcal{U}_1[r] == \mathcal{A}_t[s]$.
$\mathcal{F}_t[r] \leftarrow$ set of free neighboring vertices around robot r.
$\mathcal{F}_t[s] \leftarrow$ set of free neighboring vertices around robot s.
if $\mathcal{A}_t[r] \neq \mathcal{U}_1[s]$ **then** ▷ r follows s.
    $\mathcal{P}_t[r] \leftarrow \mathcal{A}_t[r]$, return ▷ *waiting.*
else if $\mathcal{A}_t[r] == \mathcal{U}_1[s]$ **then** ▷ r intends to swap location with s.
    if $\exists \alpha \in \mathcal{F}_t[r]$ such that $\alpha \neq \mathcal{U}_2[s]$ **then** ▷ r has the other free α.
        $\mathcal{P}_t[r] \leftarrow \alpha$, return ▷ *dodging.*
    else if $\exists \beta \in \mathcal{F}_t[s]$ such that $\beta \neq \mathcal{U}_2[r]$ **then** ▷ s has the other free β.
        $\mathcal{P}_t[r] \leftarrow \mathcal{A}_t[r]$, return ▷ *waiting.*
    end if
    if $\exists \alpha \in \mathcal{F}_t[r]$ such that $\alpha == \mathcal{U}_2[s]$ **then** ▷ r has the only free α.
        $\mathcal{P}_t[r] \leftarrow \alpha$, return ▷ *retreating.*
    else if $\exists \beta \in \mathcal{F}_t[s]$ such that $\beta == \mathcal{U}_2[r]$ **then** ▷ s has the only free β.
        $\mathcal{P}_t[r] \leftarrow \mathcal{A}_t[r]$, return ▷ *waiting.*
    end if
    if r has the other adjacent robot w **then**
        TURNING-HEAD $\leftarrow \mathcal{A}_t[w]$, and ▷ *turning-head* to new robot.
        $\mathcal{P}_t[r] \leftarrow \mathcal{A}_t[r]$, return ▷ *waiting.*
    else if s has the other adjacent robot v **then**
        $\mathcal{P}_t[r] \leftarrow \mathcal{A}_t[r]$, return ▷ *waiting.*
    end if
end if

it is blocked by the other robot v and does not have any other free neighboring vertex. In such a case, it is not r who should try to find a solution for s because s must be coordinating with v in its own decision process at the same time.

Intersections In the remaining instances (i.e., from Fig. 2(d) to 2(l)), r and s want to swap their positions, $(\mathcal{U}_1[r] == \mathcal{A}_t[s]) \bigcap (\mathcal{A}_t[r] == \mathcal{U}_1[s])$. Fig. 2(d), 2(e) and 2(f) shows T-junction or intersection configurations, where at least one robot has the other free neighboring vertex that is not the second successor vertex of the other one. Such a vertex is available for both r and s in Fig. 2(d), and only available for one of them in Fig. 2(e) and 2(f). As r believes that it has such a free vertex in Fig. 2(d) and 2(e), it can actively *dodge* itself to this vertex. For the instance in Fig. 2(f), r believes that s would apply the same strategy as what it will do in Fig. 2(e), so it just waits for r to succeed in dodging.

Narrow Corridors Fig. 2(g), 2(h) and 2(i) depict that r and s do not have the other free neighboring vertex, except for each other's second successor vertices. Such a configuration corresponds to usual narrow or dead-end corridors, where one robot has to go backwards in order to make further progress. In Fig. 2(g), each robot can go backwards, whereas only one of them can do so in Fig. 2(h) and 2(i). As r believes that it can go backwards in Fig. 2(g) and 2(h), it will

actively *retreat* itself and move backwards. When confronted with Fig. 2(i), r will take it for granted that s would retreat as what it will do in the case of Fig. 2(h), and r just needs to wait for s to make the adjustment.

Clustering Together Fig. 2(j), 2(k) and 2(l) show that r and s do not have any free neighboring vertex. In Fig. 2(j) and 2(k), r has a neighboring vertex occupied by w, while in Fig. 2(l), apart from the vertex occupied by s, r does not have any other neighboring vertex, but s has the one occupied by v. At the moment, the pairwise coordination is constructed between r and s, and they cannot make any further progress to deal with such a deadlock situation. But in Fig. 2(j) and 2(k), r can *turn* its head and reconstruct the coordination with w, instead of keeping coordinating with s. Similarly, s can reconstruct the coordination with v in its own decision process when confronted with the case of Fig. 2(l).

Algorithm 2 gives the details on how to identify which situation the robot is confronted with and which strategy it should use. A robot can employ *waiting*, *dodging*, *retreating* and *turning-head* strategies to make local adjustments in the pairwise coordination.

5 Experiments and Results

5.1 Experimental Setup

Our experimental study is performed in the Blocks World for Teams (BW4T [1]) simulator. The environment uses grid maps (see Fig. 3), and each cell in the environment only allows one robot to be present at a time. As shown in Fig. 3, the robots start from the bottom of the map, and randomly choose distinct destinations, locating at the top of the map, to navigate towards. The other gray cells represent the stationary obstacles. In the experiment, we test scalable robot teams ranging from 1 to 8, and each simulation has been run for 100 times to reduce variance and filter out random effects. Our experiments run on an Intel i7-3720QM at 2.6GHz with 8 GB of RAM.

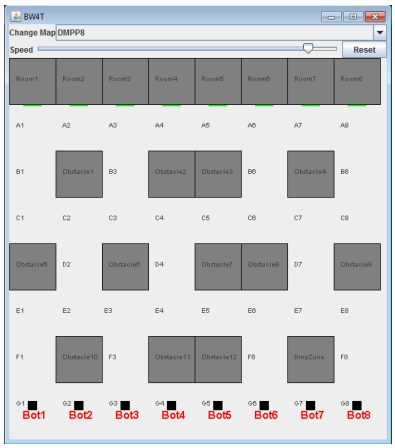


Fig. 3. The BW4T simulator.

Robots in BW4T are controlled by agents written in GOAL [19], the agent programming language that we have used for implementing our proposed approach. GOAL also facilitates explicit communication among the agents which make decisions based on its *mental states* consisting of *knowledge*, *beliefs*, and

[1] BW4T introduced in [18] has been integrated into the agent environments in GOAL [19], which can be downloaded from `http://ii.tudelft.nl/trac/goal`.

```
module pairwiseCoordination( Me, Teammate ) {
  program {
    % Me is following the trail of Teammate.
    if bel( at(Me, MyLocation), at(Teammate, TeammateLocation),
        firstSuccessor(Teammate, U1), U1 \= MyLocation ) then adopt( waiting ).
    % Me has the other free neighboring vertex.
    if bel( freeNeighboringVertex(Me, Free), secondSuccessor(Teammate, U2), U2 \= Free )
        then adopt( at(Me, Free) ) + send( allother, !at(Free) ).
    % Teammate has the other free neighboring vertex.
    if bel( freeNeighboringVertex(Teammate, Free), secondSuccessor(Me, U2), U2 \= Free )
        then adopt( waiting ).
    % Me has the only free neighboring vertex.
    if bel( freeNeighboringVertex(Me, Free), secondSuccessor(Teammate, U2), U2 == Free )
        then adopt( at(Me, Free) ) + send( allother, !at(Free) ).
    % Teammate has the only free neighboring vertex.
    if bel( freeNeighboringVertex(Teammate, Free), secondSuccessor(Me, U2), U2 == Free )
        then adopt( waiting ).
    % Me has the other neighboring robot.
    if bel( neighboringRobot(Me, OtherRobot), at(OtherRobot, Location) )
        then insert( turninghead(Location) ) + adopt( waiting ).
    % Otherwise, just wait for Teammate to turn its head.
    if true then adopt( waiting ). }}
```

Fig. 4. Implementing Algorithm 2 using GOAL agent programming language: bel(Fact) means the agent believes that Fact is true. When the agent adopt(Plan), it will execute an corresponding action to achieve Plan. When the agent insert(Fact), it will insert Fact into its belief base. Using send(allother, !at(Free)), the agent can inform the others within its coordinated network that it wants to move to the Free vertex.

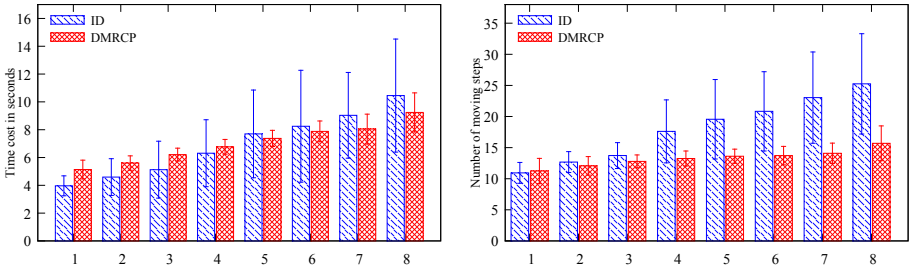
goals. Fig. 4 gives an example of how to use GOAL to implement the pairwise coordination module. Of course, the agents also need other modules to work in BW4T, such as modules to obtain environmental percepts, to handle messages, etc. Detailed information about the GOAL agent programming language and the software can be found in [19].

5.2 Results

Fig. 5 shows the results of the experimental study in which we have tested our decentralized approach, called DMRCP, and the ID approach proposed in [2]. The horizontal axis in Fig. 5 indicates the number of robots.

Time-Cost As the programs of each approach using GOAL agent programming language have different complexities, we get the general impression that for each round of decision processes, DMRCP runs slower than ID. For instance, in the single robot case, DMRCP takes 5.13 seconds, whereas ID only takes 3.92 seconds on average (see Fig. 5(a)). In the cases of 1 to 4 robots, DMRCP always costs more time than ID, but when the number of the robots increases, DMRCP takes less time than ID. Therefore, even though the running time of the DMRCP's program is more expensive than ID, DMRCP can provide a competitive solution. In addition, we can see in Fig. 5(a) that from 1 robot to 8 robots, the time-cost of ID almost linearly goes up from 3.96 to 10.45 seconds. The increasing rate is around $\frac{10.45-3.96}{8-1} \approx 0.92$. Comparatively, the time-cost of DMRCP grows

from 5.13 to 9.24 seconds, and its increasing rate is only about 0.59. This result reveals that with the increase of the robots, the time-cost of DMRCP goes up slower than ID. It is because the main strategy of ID is that if two robots have conflicting paths, one of them needs to find an alternative path, the length of which might be much longer than the previous one. It will become much more severe when the team size increases as more robots may need to frequently find alternative paths. Though DMRCP employs the waiting strategy, the robots always take the shortest paths and only need to make local adjustments, and we can see waiting may not increase time-cost in robot teams.



(a) Time-cost (b) Energy-cost

Fig. 5. Experimental results of DMRCP and ID solutions.

Energy-Cost Fig. 5(b) shows the fact that the robots take more moving steps along with the increase of the team size. Theoretically, for the single robot case, DMRCP and ID take the same moving steps as there is no other robot interfering it, which can also be seen in Fig. 5(b). With the increase of the number of the robots, the energy-cost of DMRCP goes up slowly, whereas it rises quickly in ID. For instance, in the case of 8 robots, DMRCP needs 15.69 moving steps on average; comparatively, ID takes 25.25 moving steps. The increasing rate of energy-cost in ID is $\frac{25.25-10.94}{8-1} \approx 2.04$, while it is only $\frac{15.69-11.24}{8-1} \approx 0.64$ in DMRCP. This is because DMRCP can benefit greatly from the waiting strategy as well as the local adjustments. Waiting does not increase energy-cost, and local adjustments allow the robots to take as few additional moving steps as possible.

6 Conclusions

In this paper, we analyzed the decentralized multi-robot cooperative pathfinding problem using graph-based models, and then proposed a novel fully decentralized solution to this problem. When confronted with conflicting situations, a robot only needs to coordinate with the ones locating within its coordinated network, which can reduce excessive communication in decentralized teams. Our solution allows the robots to make local adjustments by employing waiting, dodging, retreating and turning-head strategies, and the experimental results have shown that our approach provides a competitive solution to this problem.

References

1. Kaminka, G.A.: Autonomous agents research in robotics: A report from the trenches. In: 2012 AAAI Spring Symposium Series. (2012)
2. Standley, T., Korf, R.: Complete algorithms for cooperative pathfinding problems. In: Proceedings of the Twenty-Second international joint conference on Artificial Intelligence. (2011) 668–673
3. Surynek, P.: An optimization variant of multi-robot path planning is intractable. In: AAAI. (2010)
4. Desaraju, V.R., How, J.P.: Decentralized path planning for multi-agent teams with complex constraints. Autonomous Robots **32** (2012) 385–403
5. Van Den Berg, J.P., Overmars, M.H.: Prioritized motion planning for multiple robots. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). (2005) 430–435
6. Luna, R., Bekris, K.E.: Efficient and complete centralized multi-robot path planning. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE (2011) 3268–3275
7. de Wilde, B., ter Mors, A.W., Witteveen, C.: Push and rotate: cooperative multi-agent path planning. In: Proceedings of the Twelfth International Conference on Autonomous Agents and Multiagent Systems. (2013) 87–94
8. Parker, L.E.: Decision making as optimization in multi-robot teams. In: Distributed Computing and Internet Technology. Springer (2012) 35–49
9. Parker, L.E.: Current state of the art in distributed autonomous mobile robotics. In: Distributed Autonomous Robotic Systems 4. Springer (2000) 3–12
10. Silver, D.: Cooperative pathfinding. In: The First Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE). (2005) 117–122
11. Wang, K.H.C., Botea, A.: Mapp: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. Journal of Artificial Intelligence Research **42** (2011) 55–90
12. Ryan, M.R.: Exploiting subgraph structure in multi-robot path planning. Journal of Artificial Intelligence Research **31** (2008) 497–542
13. Burgard, W., Moors, M., Stachniss, C., Schneider, F.E.: Coordinated multi-robot exploration. IEEE Transactions on Robotics **21** (2005) 376–386
14. Bennewitz, M., Burgard, W., Thrun, S.: Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots. Robotics and Autonomous Systems **41** (2002) 89–99
15. Zuluaga, M., Vaughan, R.: Reducing spatial interference in robot teams by local-investment aggression. In: IEEE/RSJ International Conference on Intelligent Robots and Systems(IROS). (2005) 2798–2805
16. Dresner, K., Stone, P.: Multiagent traffic management: a reservation-based intersection control mechanism. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems. (2004) 530–537
17. Wang, K.H.C., Botea, A.: Fast and memory-efficient multi-agent pathfinding. In: International Conference on Automated Planning and Scheduling (ICAPS). (2008) 380–387
18. Johnson, M., Jonker, C., Riemsdijk, B., Feltovich, P., Bradshaw, J.: Joint activity testbed: Blocks world for teams (bw4t). In: Engineering Societies in the Agents World X. Volume 5881 of LNCS. Springer (2009) 254–256
19. Hindriks, K.: The goal agent programming language. `http://ii.tudelft.nl/trac/goal` (2013)