Asynchronous Flooding Planner for Multi-Robot Navigation

Bernd Brüggemann, Michael Brunner and Dirk Schulz
Unmanned Systems, Fraunhofer FKIE, Fraunhoferstrasse 20, Wachtberg, Germany

Keywords: Multi-Robot System, Cooperation, Planning, Graph Search.

Abstract: Within the topic of coordinated navigation of multi-robot systems (MRS) the problem may occur that there is a difference between where the robots are able to move to and where they are allowed to move. To deal with such constraints we propose a roadmap approach where those two different kinds of information are merged. So we encounter the problem of connecting several target positions with valid paths in a graph structure. Therefore, an asynchronous, parallel search algorithm is developed to find a passable and allowed way to the target positions. In addition to static planning we show that our search algorithm is able to deal with dynamic graphs and, to some extend, unknown environments.

1 INTRODUCTION

In this paper we deal with coordinated navigation of a group of robots. Our goal is to find a plan for a multi-robot system (MRS) which leads the robots to user-defined target positions without violating a given spatial constraint.

To reach this goal we use roadmap techniques to determine a collision free path for the robot system, but with special extensions to regard necessary interactions between the robots. One such interaction or, more formally spoken, spatial constraint is the permanent maintenance of communication connections. Other possible interactions or constraints include, for example, keeping a line of sight or adhering to a maximum distance between neighbouring robots. Thus, during navigation we have to consider two different kinds of information: where the robots are able to move to (*obstacle avoidance*) and where the robots are allowed to move to (*movement constraints*). In this work we merge these different kinds of information in a roadmap-like approach and provide an algorithm which finds a collision-free path and which, in addition, enables the robots to cooperate. As this is a rather simple problem in the case of only one starting and one or two end positions, we will focus on the problem of different robots having to reach different goal positions. Hence, for all following considerations we assume one starting point and several target positions.

Taking several target positions, we will show that the problem is related to the Steiner tree problem and, therefore, is np-complete. Consequently, every reasonable algorithm trying to minimize the length of the path can only be a heuristic approach. We will present such a heuristic path search algorithm which is capable of merging both types of information: where a robot is able to move to and where it is allowed to manoeuvre. Our method uses techniques from the so-called MPR (multi-point relay) Flooding, which has its origins in the network communication community, in order to minimize the number of nodes in the solution. Thereby, we interpret each node in the roadmap as an independent entity, i.e. an agent. This enables the algorithm to perform a parallel search on the whole environment. Additionally, due to special characteristics of this approach, we are able to deal with dynamic environments: if the roadmap changes and a plan gets outdated, the algorithm successfully handles that and re-establishes the multi-robot plan.

The remainder of the paper is organized as follows: in section 2 we present some relevant work on coordinated navigation in multi-robot systems and give a short introduction to the MPR Flooding problem. Section 3 explains the basic data structures we use for the roadmap approach. It provides details about the information merging process and shows the relation between our problem and the Steiner tree problem. After that the proposed agent-based flooding algorithm is described in subsections 3.2 and 3.3, its behaviour on dynamic graphs is shown in chapter 4. Finally, a conclusion and a short outlook are given in chapter 5.

Brüeggemann B., Brunner M. and Schulz D..
Asynchronous Flooding Planner for Multi-Robot Navigation.
DOI: 10.5220/0004457802220230
In *Proceedings of the 10th International Conference on Informatics in Control, Automation and Robotics* (ICINCO-2013), pages 222-230
ISBN: 978-989-8565-71-6

2 RELATED WORK

Navigating a multi-robot system (MRS) with respect to various spatial constraints is a well-known problem in robotics. However, most papers deal with some kind of communication constraint. This is, of course, an important constraint for MRS because many cooperating tasks assume steady communication between the robots. But a broader view on such spatial constraints is necessary to cover a larger field of possible applications.

The planning problem addressed in this work has some similarities with the multi-robot routing problem described in (Lagoudakis et al., 2005). There, the problem of assigning the targets to the robots is examined, which is already the main difference between both problems. While in the multi-robot routing problem the target positions have to be visited once, in our case a robot has to reach a goal point and stay there till the end of the plan. Mosteo et al. added the communication constraint to the multi-robot routing problem. In (Mosteo et al., 2008) and (Mosteo et al., 2009) the communication constraint is viewed as a local constraint between each robot. In their work the authors use a reactive approach: whenever the signal strength between two robots drops below a certain threshold, the distance between them cannot be further extended. The communication is modelled as a spring-damper system, leading to the typical behaviour: if a robot proceeds, it drags a line of robots along to ensure that always relay robots are available for communication with the control station. Such a reactive approach always holds the risk of suddenly losing communication, for example, if the leading robot vanishes behind a massive obstacle.

Our algorithm was mainly inspired by the MPR Flooding algorithm as proposed for ad-hoc network protocols in (Com, 1996). The aim of the MPR Flooding is to identify a set of communication nodes which covers the whole communication network. So, if all MPR nodes repeat a message, each node receives that message at least once. A greedy algorithm to obtain such a set of nodes as well as some improvements are described in (Qayyum et al., 2002). However, finding an optimal set of MPR nodes is np-complete, as is the general problem of an energy-efficient flooding of a network (see (Čagalj et al., 2002)). Second, our approach is based on the idea of using agents for multi-robot planning problems. Solutions for CSP (constraint satisfaction problem) and especially for the distributed version, DisCSP, can be used, for example, in multi-robot exploration as shown in (Monier et al., 2010). Therefore, our algorithm shows some similarity with the search algorithm shown for example in (Zou and Choueiry, 2003). In contrast to such algorithms, we model the spatial constraint in our algorithm by the limited communication between the agents.

In contrast to the mentioned solutions for the multi-robot routing problem, here we present a planning algorithm which defines its actions before the execution. Most coordination algorithms in literature use a reactive approach to handle constraints and to react if they might be violated. But doing so, the class of addressable constraints is limited to continuous constraints. If the connections created regarding a non-continuous constraint are abruptly aborted, a reactive method can hardly deal with that. Our approach, in contrast, offers the operator a global plan, which obeys the constraint, and can, in the dynamic case, react to changes in the environment.

3 AGENT-BASED FLOODING SEARCH

3.1 Basics

As stated in the beginning, our basic goal is to find a plan which leads a multi-robot system from a common start to several user-given target positions. During the execution of the plan, we have to obey a given spatial constraint. So the problem can be formalized as follows:

- Given:
  - n robots and m target positions; $n > m$
  - A spatial constraint C
- Search:
  - A configuration of the MRS witch includes at least all target positions and which fulfills the constraint
  - A path to this configuration which continuously obeys the constraint

In this work we use a roadmap approach to represent where a robot is able to move to. The resulting graph is called the *movement graph* $G_{mov} = \{V_{mov}, E_{mov}\}$. Its nodes V_{mov} are evenly distributed over the environment. Between two neighbouring nodes there is an edge $e_i \in E_{mov}$ if there is no obstacle in between and the slope is not too steep. We allow horizontal, vertical and diagonal edges. The weight of e_i is equal to the Euclidean distance between the nodes.

To coordinate the robots, we look at spatial information like the maintenance of communication or up-keeping the line-of-sight. Those constraints can be formulated as $C(v_i, v_j) = \{0,1\}$ with $v_i, v_j \in V_{mov}$.

(a) (b) (c)

Figure 1: The basic SCG idea: possible movements are merged with the allowed area. This results in a list of valid movements with respect to the position of a robot. (a) The movement graph is represented by the streets which a robot can traverse. (b) The given spatial constraint (here, e.g. communication range) is valid within the coloured area. (c) Merging the possible movements and the allowed area results in a choice of streets on which another robot may move without violating the given constraint.

So it indicates if the constraint is fulfilled between two nodes ($C(v_i, v_j) = 1$) or not ($C(v_i, v_j) = 0$). This results in the constraint graph $G_{con} = \{V_{mov}, E_{con}\}$ which consists of the same nodes as the movement graph. Between two nodes v_i, v_j from V_{mov} there is an edge $e_i \in E_{con}$ if $C(v_i, v_j) = 1$. The weight of e_i is 1.

At this point we have the information if a robot is able to move to a certain position and the information if it is allowed to move there. To find a way the robot is able to follow, we have to find a way from the starting node to the target node in G_{mov}. But to judge if a movement of a robot is allowed, we have to take the positions of all robots into account. As long as all robots are directly connected in the constraint graph, the current robot configuration is allowed. All robot positions represent one connected component in G_{con}. Thus, before performing a step with a robot, one has to test if there remains a connected component afterwards.

Obviously, this leads to a large number of necessary tests needed between G_{mov} and G_{con}. To reduce the number of tests we propose the Separated Connection Graphs (SCG). Each SCG results from a merging operation of G_{mov} and G_{con} at one node v_i. $\mathrm{SCG}(v_i) = \{V_{SCG(v_i)}, E_{SCG(v_i)}\}$ consists of a set of nodes $V_{SCG(v_i)}$ which includes v_i and each node which is a direct neighbour to v_i in G_{con}. There is an edge $e \in E_{SCG(v_i)}$ between v_i and $v_s \in V_{SCG(v_i)}$, if v_i and v_s are within one connected component in the subgraph $G_{sub} = \{V_{SCG(v_i)}, E_{sub} \subseteq E_{mov}\}$. Thus, if one robot stands at v_i, $\mathrm{SCG}(v_i)$ represents a look-up table to which node another robot is allowed and able to drive. The idea behind the SCGs can be seen in figure 1.

Using the SCGs it is possible to find a valid path from a starting node v_s to a target node v_t. This path can be described as a sequence of SCGs: $\mathrm{SCG}(v_s)$, $\mathrm{SCG}(v_i)$, $\mathrm{SCG}(v_{i+1})$, ..., $\mathrm{SCG}(v_t)$. When travelling along such a path, one robot has to be left at each node v_i, v_{i+1}, ... to satisfy the constraint. We call these nodes *relay nodes*, as the robots on that nodes are working as a sort of relay stations. Note that in spite of the term 'relay node' the actually used spatial constraint can be freely chosen; it is not necessarily communication range.

From such a path to a target node one can get an idea how a complete plan for a multi-robot systems looks like: a starting node, several additional relay nodes and the different target nodes are connected by paths which are allowed and traversable by the robots. It is important to mention that such a plan does not yet assign an executable path or a particular node to any robot. In fact, the actual execution of such a precomputed path with a real multi-robot system requires further considerations and is far beyond the scope of this paper. The interested reader can find additional information in (Brüggemann et al., 2012).

As we need a robot for each relay node, an obvious optimization is to minimize the number of relay nodes. This raises the question how hard it is to find an optimal plan for a set of target nodes. As already mentioned, a relationship to the well-known Steiner tree problem (see (Gilbert and Pollak, 1968)) can be shown. To be more precise, finding a minimal distribution of relay nodes which connects all target nodes and the starting node is equivalent to the Steiner tree problem:

Given a graph $G = \{V, E\}$ and a set of terminals $T \subseteq V$, a Steiner tree (sometimes also called minimal Steiner tree) is a graph $G_{st} = \{V_{st}, E_{st}\} \subseteq G$ with $T \subseteq V_{st}$ and the weight of $|E_{st}|$ minimal. Exact solutions for this problem are known to be np-complete.

Be V_{t_i} the set of target nodes and v_s the starting node. We are searching for a minimal set of relay nodes V_{r_i}, such that the set of nodes $V_{t_i} \cap v_s \cap V_{r_i}$ are connected in G_{con}. The edges in G_{con} have a weight of 1, the number of nodes is equal to the number of edges $+1$. Thus, we are searching for a set of nodes which minimizes the total weight of the edges. With $V_{t_i} \cap v_s = T$ this is equivalent to the Steiner tree problem.

Consequently, every algorithm which provides a set of relay robots or a plan in reasonable time can be only heuristic.

Our Agent-Based Flooding Search is such a heuristic approach. It is based on a flooding mechanism similar to the MPR flooding combined with an agent-based search. Based on the definition for G_{mov} and G_{con}, each node is taken as a separate entity with an internal state consisting of the 3-tuple {active | inactive, expanded | not expanded, priority}. Each agent is initialised as {inactive, not expanded, ∞}.

Each agent is able to communicate with all other agents in its neighbourhood. To gather plans which respect the movement graph as well as the constraint graph we define the neighbourhood of an agent as follows: two agents a_i and a_j are neighbours if a_i and a_j are connected in SCG(a_i) as well as in SCG(a_j). An agent does not have to be active to communicate. Communicating agents are able to mutually exchange the following information resp. commands:

- An active agent can activate an inactive agent.
- An agent informs its neighbours that it is currently expanding.
- An agent receives information about its direct neighbourhood, including the internal state of each neighbour.
- An agent receives the same information about each neighbour's neighbourhood.

With these basic foundations we are now able to define an undirected flooding search algorithm. In the first step, the *expansion step*, agents start to activate other agents, resulting in an undirected search for possible connections to other activated agents. The next section describes this step in detail. Since after the end of the expansion step the resulting network of activated agents contains far too many nodes, a further *optimization step* is necessary. This step is then described in section 3.3.

3.2 Expansion Step

The expansion step is similar to the MPR flooding algorithm. First, those agents representing the starting position and the end positions are activated and their priority is set to zero. The expansion step lasts until all activated agents build up one connected component in the constraint graph.

Each *active, not expanded* agent a_i constantly checks if there is another *active, not expanded* agent with a priority lower than its priority. If there is no such agent, a_i starts to expand with broadcasting a *currently expanding* message to its neighbours. This prevents other agents with the same priority from expanding.

Now, the agent a_i gets its 1-hop neighbourhood of non-active agents. Let be H_1 this set of agents. Now a_i gathers the 1-hop neighbourhood of each agent in H_1. Let be H_2 those agents. We build H_2 in a way that it will not contain any agents from H_1. Now we search for $H_1^{sub} \subseteq H_1$ so that those agents can communicate with all agents in the set H_2. All agents in H_1^{sub} will now get activated and receive a priority which is the priority of $a_i + 1$.

Choosing the set H_1^{sub} can be done by a modified greedy strategy. First, we identify the set of agents S_{alone} in H_2 which can only be addressed by a single agent in H_1. Every agent in H_1 which reaches an agent of S_{alone} must be activated. Let those newly activated agents be in the set S_{single}. Now all agents addressed by S_{single} can be removed from H_2 and all activated agents are removed from H_1. Then we search for the agent a_j in H_1 which can reach the most agents in H_2. a_j is activated and removed from H_1. All agents connected to a_j in H_2 are also removed. This is repeated until H_2 is empty.

During the expansion step a central component of the algorithm continuously checks whether all activated agents build one connected component in G_{con}. We call this the *breaking condition*. As soon as this condition becomes true for the first time, the expansion step is stopped immediately and no agent is allowed to expand any more. At this time, the network of activated agents represents a valid solution to the multi-robot path planning problem but includes many unnecessary nodes. Thus, the Agent-Based Flooding Search continues with an additional *optimisation step*.

3.3 Optimisation Step

For the optimisation step we will deal with the graph $G_{mpr} = \{A_{active}, E_{active}\}$ which is generated from the active agents' network. Let be A_{active} all active agents and E_{active} all edges representing that two agents in A_{active} are neighbours. This graph G_{mpr} is, generally spoken, a valid plan for the multi-robot system, as it connects the starting point with some relay points and the target positions. Additionally, due to the definition of the SCGs, the robots are able and allowed to pass from one relay point to another. But normally G_{mpr} contains too many relay positions, most of which are unnecessary. To reduce the number of nodes the following optimisation step was implemented.

As it is known from the similarity to the Steiner tree problem, a good solution will be a tree. Therefore, we use a minimal spanning tree (MST) algorithm on G_{mpr}. In the resulting tree T_{mpr} we can check each leaf if it represents either the starting position or a target position. If it is not, it represents a relay node.

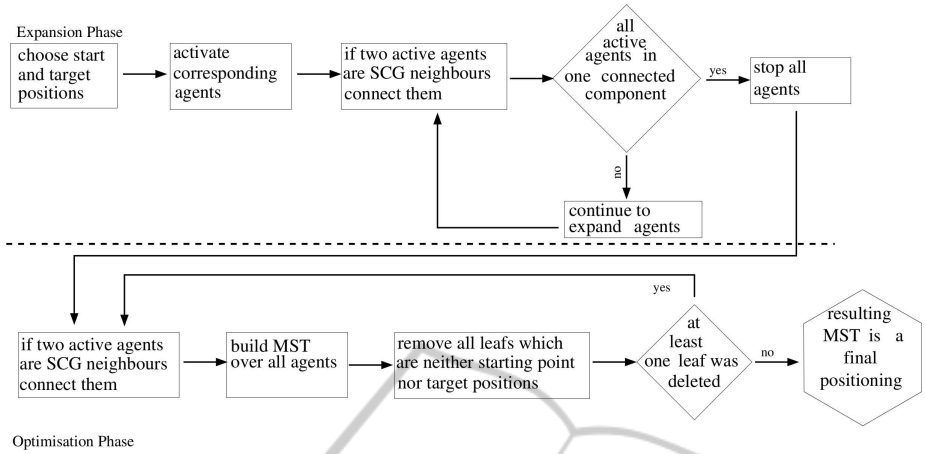


Figure 2: Flowchart of the Agent-based Flooding Search algorithm.

Algorithm 1: Agent-based flooding planner.

while all agents are not within one connected component **do**
  for Each active, not expanded agent **do**
    while Neighbouring agent is expanding **do**
      WAIT
    end while
    Let be $\mathcal{L}_1$ the set of agents in 1-hop neighbourhood which are not active
    Let be $\mathcal{L}_2$ a list of all non-active agents in 2-hop neighbourhood which are not in $\mathcal{L}_1$
    while $\mathcal{L}_2$ is not empty **do**
      Activate agent c in $\mathcal{L}_1$ which is connected to the most agents in $\mathcal{L}_2$
      Remove all agents reach by c in $\mathcal{L}_2$
      Remove c from $\mathcal{L}_1$
    end while
    Mark this agent as expanded
  end for
end while
Let G_{mpr} include all active agents and their edges
while at least one leaf is removed from G_{mpr} **do**
  Let be G_{mst} = minimal spanning tree of G_{mpr}
  Remove all leafs from G_{mst} in G_{mpr} which represent no target positions
end while

However, as a leaf it is not necessary for the plan and can be deleted. After this deletion the remaining nodes are once again connected if the corresponding agents are neighbours. With this smaller graph the optimisation step is repeated as long as no further unnecessary node can be found. The resulting MST represents the desired plan for the multi-robot system.

A flowchart of the complete Agent-Based Flooding Search algorithm can be seen in Figure 2, and Figure 3 pictures a complete example of the planning process.

3.4 Characteristics and Evaluation

An advantage of the presented search algorithm is its completeness. If there is a solution it will find one; if there is no solution, it will report it. In the case of an unsolvable configuration, the search algorithm will not move from the expansion step to the optimization step. The breaking condition, saying that all agents are in one connected component, can never be reached because that connected component would be a valid (yet suboptimal) plan. Thus, at a certain point, there are no more active agents which are not expanded. At this point we know that there is no valid solution.

Another advantage of the agent-based search is that the agents just need local information about which agent has to be expanded, enabling the expansion step to be parallelised. Hence, each agent can be expanded independently. This works well for regions far away from each other. But if those regions grow together, it might happen that a certain string of agents expands faster than other agents as the expansion is done in an asynchronous way. This may increase the number of relay nodes in the resulting plan. The priority values in the agents' internal states help to balance the expansion among the different agents. As a result, the search process is more evenly distributed over the target area, especially if two expanding regions grow together. Please note that the Agent-Based Flooding Search is usable as an asynchronous, parallel search algorithm, but cannot be used in a decentralized way as the breaking condition has to be checked by a central entity.

As we have shown, the problem we try to solve

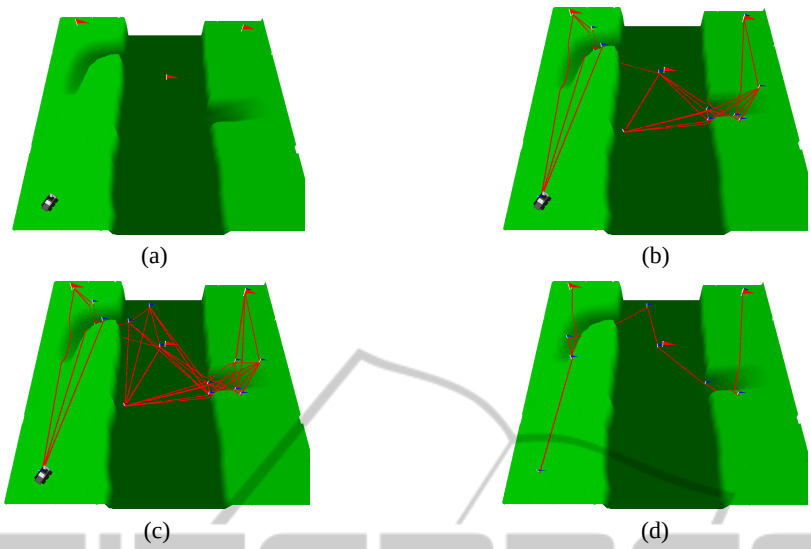


Figure 3: An example of the Agent-Based Flooding Search for multi-robot planning. (a) Shows the initial state. The robot represents the starting position, the red flags the target positions. (b) The search during an expansion step. (c) Here the breaking condition triggers. All agents are within one connected component. (d) The resulting tree which connects the starting position and the target positions.

is equivalent to the Steiner tree problem. Thus we compare our Agent-Based Flooding Search with a well-known Steiner tree heuristic, Melhorn's algorithm (see (Mehlhorn, 1988)). As we have to find the optimal number of relay nodes, we use Melhorn's algorithm only on G_{con} while the Agent-Based Flooding Search works on the SCGs and therefore considers G_{con} as well as G_{mov}. Additional benefits of this approach for multi-robot planning can be seen in (Brueggemann et al., 2012). To compare these two algorithms, we used four different simulated environments and three different constraints. Two of the environments (landscape and canyon) are artificial environments while the other two (Geb3 and TDSuS) are real world maps gathered by a 3D laser scanner. The three constraints are:

- distance constraint: $C_{dist}(v_i, v_j) = 1$ if the Euclidean distance between v_i and v_j is lower than a certain threshold.
- communication constraint: $C_{com}(v_i, v_j) = 1$ if a given wave propagation model states that there is communication available between v_i and v_j.
- visibility constraint: $C_{vis}(v_i, v_j) = 1$ if a robot on v_i is able to see another robot on v_j and vice versa.

For each combination of environment and constraint we performed 500 runs in simulation. Each of those runs had a different configuration of starting position and target positions. The number and placement of starting position and end positions were randomly chosen. There were always one starting position and between 3 and 10 end positions, randomly distributed in the environment. Only solvable configurations were taken into account.

Table 1: Results of the simulation tests. For each planner, each map was tested with each constraint in 500 runs.

	Robot in final positioning			Max length in plan			Time for final positioning			Total planning time		
map	vis	com	dist	vis	com	dist	vis	com	dist	vis	com	dist
Steiner Tree heuristic												
Canyon	7.1	8.7	7.7	57.9	63.3	47.8	54.0	13.0	27.1	57.7	14.4	28.5
Landscape	7.3	8.2	7.5	39.8	38.1	33.6	56.0	16.0	26.1	59.9	16.8	27.7
Geb3	7.4	7.4	6.8	27.2	28.8	25.1	24.5	7.8	14.1	24.7	8.2	14.7
TDSuS	8.6	7.9	6.9	64.3	49.3	37.3	16.8	29.9	74.8	18.6	31.2	77.9
AgentPlanner												
Canyon	8.1	8.3	8.5	37.6	45.3	38.2	15.1	32.0	79.0	18.2	32.5	80
Landscape	7.5	8.9	8.2	29.7	28.8	28.7	8.2	12.4	18.4	9.7	13.0	19.5
Geb3	8.3	7.7	7.7	22.8	24.6	25.5	2.0	12.2	22.7	2.4	12.6	23.5
TDSuS	10.3	8.5	7.9	36.7	37.2	35.9	29.7	19.0	60.3	30.1	20.2	63.2

Several different characteristics of solutions from the Agent-Based Flooding Search in comparison with the Steiner Tree heuristic as baseline can be seen in table 1.

These numbers provide a strong indication that, regarding performance, the Agent-Based Flooding Search is comparable to Mehlhorn's Algorithm, at least for our typical application environments. The number of robots in the final positions is similar in the Steiner Tree heuristic and in the agent-based planner, which, depending on the environment, uses not more than one additional robot. On the other hand, the agent-based planer computes plans with a much shorter maximum length of one particular path, which significantly reduces the execution time for such plans. Generally, the computation time of the two approaches depends strongly on the environment and shows no clear advantage for one specific approach.

4 SEARCH ON DYNAMIC GRAPHS

So far we have considered static graphs. As already mentioned, an important advantage of the Agent-Based Flooding Search is its ability to deal with dynamic graphs. In general, dynamic graphs can change their edges or nodes. In this paper we will address two different kinds of dynamic environments. In the first part we deal with a known environment which might change after a solution is found. The second subsection deals with unknown environments. In this setup information about the environment is collected while the robots are moving, and the representing graphs are build up online.

4.1 Dynamic Environments

In dynamic environments we just address the removal of edges in G_{mov} or G_{con}. We do not deal with changes due to new edges. They might offer the opportunity to enable shorter plans but will not corrupt an existing path. So only the removal of one or more edges may corrupt the plan in a way that requires re-planning.

To deal with such a dynamic environment we have to change the Agent-Based Flooding Search so that it will react to changes in the graphs. Simply speaking, the central entity which checks the braking condition in the expansion step (see section 3.2) continues to run during the plan execution phase. Thus, as soon as the graph changed in a way that the activated agents are no longer in a connected component, this is noticed and the search algorithm restarts with the expansion step. Thereby, the search process does not have to be completely re-initialised. Instead, as there are already some activated agents from the former solution, the search can be restarted with a predetermined direction. If there are only some small changes necessary, the search process will find them quickly. However, the completeness guarantees a valid solution even if the former plan does not fit at all. This makes the Agent-Based Flooding Search a valid approach for dynamic graphs. An example re-planning due to a change in the environment can be seen in Figure 4:

Here the robot represents the starting position; the red flags represent the target positions; the blue flags represent the relay nodes. The red lines show the Steiner tree-like solution for connecting the starting position with the target positions. The turquoise line shows the path the robots are going to follow. In Figure 4a the existing solution is invalid because a blocking obstacle appeared on the lower bridge. As now there is no longer a way between the starting position and the relay position in the upper right, the representing agents are no neighbours (see figure 4b). Thus the expansion step starts again. Figure 4c shows the new expansion, which lasts until all active agents are connected again. As the originally activated agents are also used, the search can be performed with previous knowledge. A new valid solution is shown in figure 4d. Now the path follows the upper bridge. All changes are localized in the region where the obstacle occurred, the other half of the plan remains untouched.

4.2 Unknown Environments

In an unknown environment the status of G_{mov} and G_{con} is unknown from the beginning. We assume that only the area a robot has already visited is known and does not change any more. So the movement graph and the constraint graph will get known only during the execution of the plan (see Figure 5). Thus the plan will have to change over time.

To find a plan with the Agent-Based Flooding Search we have to make some assumptions about the *unknown areas*. The frontier between known and unknown areas is continuous. This is necessary because otherwise the movement graph would not reach into unknown environment. There are no obstacles in unknown areas and the surface is flat. This means the movement graph in unknown areas is fully linked and constraints are not influenced by the environment. And if there is only one node of v_i, v_j in $C(v_i, v_j)$ inside the unknown area, we assume the same height for both nodes.

Those assumptions enable the Agent-Based Flooding Search to find a plan and update it while the robots are moving. In simulated environments we could observe that the plan as a whole often changes drastically during execution. This significantly slows down the execution time because every time the robots have to be called back. Therefore, we examined three methods to stabilize the plans: Firstly, we penalized paths through unknown areas. This is a common method to avoid alternating paths during exploration. Secondly, we tried to reuse relay nodes already settled by a robot. As soon as a relay node is occupied by a robot, this relay node has to be included in any further solution. Such a node can only be deleted if it is a leaf in the plan. Thirdly, we relaxed the primary assumptions of unknown areas. They are very conservative as they usually underestimate the number of edges in G_{con}. The relaxing method, in contrast, always assumes a connection between two nodes v_i and v_j in G_{con} if the line from v_i to v_j cuts unknown areas.

To evaluate those methods we have run different

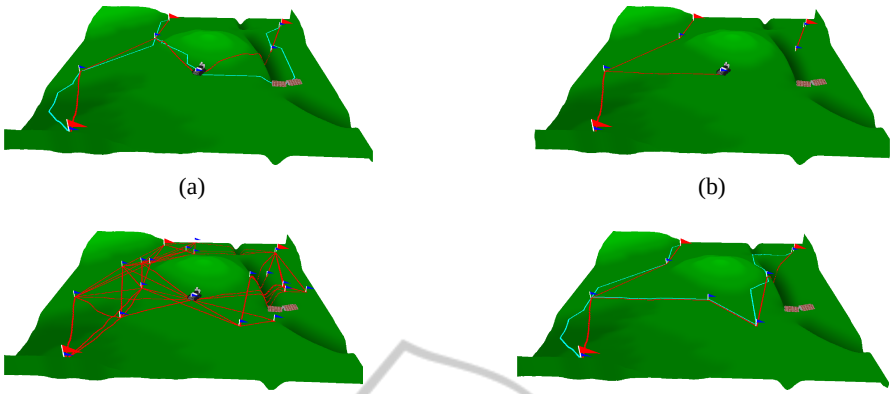


Figure 4: A planning example in a dynamic graph.

configurations of target positions in different environments with different constraints. We counted the number of steps the robots performed until the plan is solved and normalized the results to the number of steps needed by the original algorithm. Table 2 shows the results.

Table 2: Influence of stabilizing methods on execution time. The execution time of the original algorithm was used to scale the results.

	Original algorithm	Reuse relay nodes	Penalize unknown paths	Relaxing constraints
Distance constraint				
Canyon	1	1.05	0.63	1.33
Landscape	1	3	0.96	0.96
Communication constraint				
Canyon	1	0.68	1,47	0.62
Landscape	1	0.47	1.17	0.43
Visibility constraint				
Canyon	no solution	no solution	no solution	solution
Landscape	1	0.59	1.84	0.28

One can see that the reuse of occupied relay nodes and the relaxing method have a positive influence on the execution time. However, due to the way the reuse method works, the number of robots needed to execute the plan always rises during execution. The relaxing method on the other hand does not need more robots than the original algorithm. Additionally, the relaxing method allows solutions in special cases (see table 2, visibility constraint, canyon). Here the robots have to use ramps to get to different sections of the environment. Such a ramp is not passable for the original algorithm. It is most likely that the direct connection between a node v_i on the ramp in the known area and v_j in the unknown area will cut the surface, as the original algorithm assumed that v_i and v_j are on the same hight. This results in a violation of the visibility constraint and thus in a non-traversable edge. So the relaxing method enlarged the solution space in a way that in special environments a solution can be found.

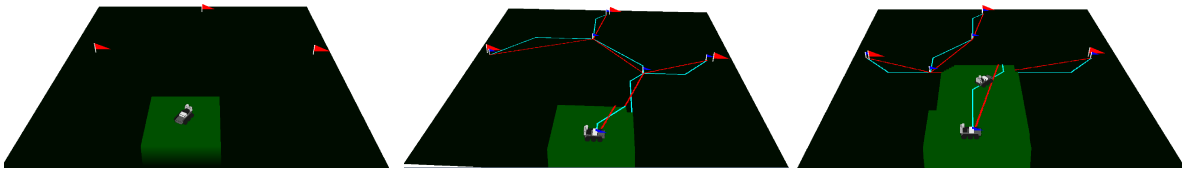
Nevertheless, even with the relaxing method, the execution time of plans in unknown environments is high. This results from the two opposing tasks the robots have to perform in an unknown environment: 1) reach the target positions as fast as possible and 2) explore the environment to find a good plan. So it might be a good idea to combine the search with multi-robot exploration like in (Burgard et al., 2000)). Nevertheless, the Agent-Based Flooding Search is also able to deal with unknown environments.

5 CONCLUSIONS AND FUTURE WORK

In this paper we presented a search algorithm which is able to find short routes from one starting position to several target positions and, therefrom, calculates navigation plans for a multi-robot system. The presented approach uses methods from ad-hoc network implementations and merges them with an agent-based view, resulting in an asynchronous parallel undirected search. We have also shown that the algorithm works on known dynamic graphs and, to a somewhat lesser extent, in unknown graphs.

As the presented search uses only local information about the neighbourhood of the activated agents, the Agent-Based Flooding Search does not have to be changed for using it on dynamic graphs. Although there are solutions for dynamic Steiner trees (see (Ding and Ishii, 2000) or (Blin et al., 2009)) our solution has the advantage of not having to deal with dynamic graphs as a special case. The use of only local information also reduces the computation time in large environments, if only small parts of it are really affected.

Our main application for the Agent-Based Flooding Search is to compute navigation plans for a multi-robot system where the problem is to maintain a



(a) (b) (c)

Figure 5: The first steps of a planning query in an unknown environment. (a) Initially, only the area around the starting point is known. (b) A first plan is generated by the search algorithm. (c) During the execution of the plan, more and more of the environment is discovered. So the initial plan has to be adjusted.

certain constraint, like continuous communication. Therefore, we not only have to find a valid way for each robot, but we also have to find relay positions to hold the constraint. The actual execution of a computed plan using a real heterogeneous multi-robot system is explicitly described in (Brüggemann et al., 2012). In the next step we want to find new applications where such a search can be useful. This might be applications in which we have to search for several targets in a large, dynamic graph. Especially if the neighbourhood is easy to compute, our Agent-Based Flooding Search might add some value to such problems.

REFERENCES

(1996). Radio equipment and systems: High performance radio local area network (hiperlan) type 1, functional specifications,. Technical report, ETSI STC-RES10 Committee.

Blin, L., Potop-Butucaru, M., and Rovedakis, S. (2009). A superstabilizing log (n)-approximation algorithm for dynamic steiner trees. *Stabilization, Safety, and Security of Distributed Systems*, pages 133–148.

Brueggemann, B., Brunner, M., and Schulz, D. (2012). Spatially constrained coordinated navigation for a multi-robot system. *Ad Hoc Networks*, (0):–.

Brüggemann, B., Langetepe, E., Lenerz, A., and Schulz, D. (2012). From a multi-robot global plan to single robot actions. In *Proceedings of Informatics in Control, Automation and Robotics (ICINCO)*.

Burgard, W., Moors, M., Fox, D., Simmons, R., and Thrun, S. (2000). Collaborative multi-robot exploration. volume 1, pages 476–481 vol.1.

Ding, S. and Ishii, N. (2000). An online genetic algorithm for dynamic steiner tree problem. In *Industrial Electronics Society, 2000. IECON 2000. 26th Annual Confjerence of the IEEE*, volume 2, pages 812–817 vol.2.

Gilbert, E. and Pollak, H. (1968). Steiner minimal trees. *SIAM Journal on Applied Mathematics*, pages 1–29.

Lagoudakis, M. G., Markakis, E., Kempe, D., Keskinocak, P., Kleywegt, A., Koenig, S., Tovey, C., Meyerson, A., and Jain, S. (2005). Auction-based multi-robot routing. In *Robotics: Science and Systems*, pages 343–350. Citeseer.

Mehlhorn, K. (1988). A faster approximation algorithm for the steiner problem in graphs. *Inf. Process. Lett.*, 27(3):125–128.

Monier, P., Doniec, A., Piechowiak, S., and Mandiau, R. (2010). Metrics for the evaluation of discsp: some experiments on multi-robot exploration. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*, volume 2, pages 370–373. IEEE.

Mosteo, A. R., Montano, L., and Lagoudakis, M. G. (2008). Multi-robot routing under limited communication range. In *IEEE International Conference on Robotics and Automation (ICRA) 2008*.

Mosteo, A. R., Montano, L., and Lagoudakis, M. G. (2009). Guaranteed-performance multi-robot routing under limited communication range. In *Distributed Autonomous Robotic Systems 8*, pages 491–502. Springer Berlin Heidelberg.

Qayyum, A., Viennot, L., and Laouiti, A. (2002). Multipoint relaying for flooding broadcast messages in mobile wireless networks. In *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, pages 3866–3875.

Čagalj, M., Hubaux, J.-P., and Enz, C. (2002). Minimum-energy broadcast in all-wireless networks: Np-completeness and distribution issues. In *Proceedings of the 8th annual international conference on Mobile computing and networking*, MobiCom '02, pages 172–182, New York, NY, USA. ACM.

Zou, H. and Choueiry, B. (2003). Multi-agent based search versus local search and backtrack search for solving tight csps: A practical case study. In *Working Notes of the Workshop on Stochastic Search Algorithms (IJCAI 03), Acapulco, Mexico*, pages 17–24.