Combining the Multi-Agent Flood Algorithm with Frontier-Based Exploration in Search & Rescue Applications

Florian Blatt and Helena Szczerbicka
Simulation and Modelling Group
Faculty of Electrical Engineering and Computer Science
Leibniz University Hannover, Germany
Telephone: +49 511 762 3396
Fax: +49 511 762 3675
{blatt,hsz}@sim.uni-hannover.de

***Abstract*—The Multi-Agent Flood algorithm is an algorithm to control multiple autonomous agents to explore unknown terrain and to find points of interest in the newly uncovered territory. It does so by using a mix of indirect and direct communication. The direct communication via radio signals allows for a faster exchange of data between the agents. This layered communication model allows for a robust data transfer that is needed in SAR scenarios, as the algorithm can still use the indirect part of the model if the direct part is not available anymore. Currently the exploration done by the agents is decided randomly without any intelligent input based on the gathered data. The agents will simply try to move along a line until they will hit an obstacle or a point of interest is found. This work introduces ways to steer the autonomous agents based on the collected data, additionally this data will be used to decide which part of the map should be explored next. Another optimization of the performance is also the increased cooperation between the agents, as the agents will also exchange these destination points, thus reducing the number of agents which will explore the same part of the area, leading to a greater spread of the agents through the terrain.**

Keywords: *search and rescue, multi-agent systems, communication, cooperation, frontier detection*

I. Introduction

Current search and rescue robots that were and are used at various disaster sites still have to be operated by human team members. Additionally these robots usually are connected by a tether to the base camp, as this allows for an error free data transmission between the operator and the robot, c.f. [1], [2], [3], [4]. An alternative to this design would be the use of one or more autonomous robots acting as a team, without any human interference or controlling. This need for control lead to the introduction of multi-agent systems for search & rescue missions, allowing a number of autonomous robots to act in concert to explore the unknown terrain of the disaster site and find points of interest or in this case human victims, [5], [6], [7], [8], [9].

Our previous works have introduced the Multi-Agent Flood algorithm as a viable alternative for a multi-agent system in search and rescue scenarios. We also presented a robust communication model, that is based on indirect and direct data transfer between the agent, which allows for a robust link amid all the agents and thus the possible creation of ad-hoc networks. Another optimization recommended in our work was a way to disperse RFID tags in the environment to act as data storage for the indirect part of the communication, compare respectively [9], [10], [11]. But there are still potential options available which can be optimized. All of the aforementioned multi-agent systems do not use a goal oriented exploration. Usually the agents choose a random direction and follow that direction until they arrive at an obstacle or they encounter a point of interest. Additionally the movement may be influenced by markings in the terrain or otherwise collected data. Some of the previously mentioned algorithms ([6], [7], [9]) for example use the markings to mark already visited terrain and thus the agents try to stay clear of this part of the area. This is a possible way to force the agents to move towards parts of the terrain that are not yet marked. A first step to improve this behavior would be to specify target points for the agents, so that they will be guided to specific points in the area and in turn speed up the exploration of the whole map.

This improvement would also reduce one kind of error introduced by the previous behavior of the agents. As mentioned above the agents will mark already visited terrain and will try to ignore this terrain. Due to this behavior the markings can prevent the agents from exploring some parts of the terrain. An example of this can be seen in Fig. 1. The agent, represented by the gray rectangle, will avoid the room as the blue markings in front of the door will signal that the area at the entrance of the room was already explored, although the rest of the room is still unexplored. In the worst case this can lead to the exploration of the rest of the terrain while leaving some parts still unexplored, although these parts may be near the starting point.

The next point of improvement would be the navigation towards these specific points. This is an already known problem to find a viable path between two points, in this case the current position of the agent and the specified point of exploration. Examples of these algorithms are the well known A* algorithm,

SummerSim-SPECTS, 2017 July 9-12, Bellevue, Washington, USA;

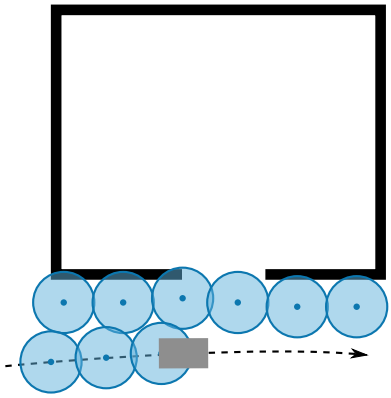

Fig. 1. Markings cutting of unexplored terrain

the Jump Point Search algorithm, or to keep things simple the Bug algorithms from Lumelsky, see respectively [12], [13], [14].

To further increase the spread of the agents in the unknown terrain and thus leading to a faster exploration the cooperation between the agents has also to be raised. Currently the agents of the MAF algorithm are able to exchange the collected map data as long as they are able to communicate directly with each other using wireless communication. After exchanging this data between themselves the agents should now also be able to compare their exploration targets and in turn change their target, if the other agent is also traveling to the same target. This will decrease the number of agents that are trying to uncover the same parts of the area.

The following section will contain an overview of the current State of the Art. In the third section a short description will follow on how the various algorithms were combined to form the different improvements to the Multi-Agent Flood algorithm. The next section, Section IV, will expand on how the experimental simulations to evaluate these adaptions were set up. The results of these simulations will be discussed in the fifth section followed by a conclusion in the last section of this paper.

II. State of the Art

A. The Multi-Agent Flood algorithm

The Multi-Agent Flood algorithm (MAF) is used to explore unknown terrain in order to find points of interest. It is designed as a multi-agent system [9], [10], [11]. The algorithm is based on two modes: the search and the return mode. In the search mode the agents explore the unknown terrain and drop markings in the terrain at specific intervals. These markings contain the current distance at the time of the drop from the specific agent towards the starting point. If an agent encounters a marking with a higher distance than its own distance, it overwrites the data stored in the marking with the lower value. The stored distance and the dropped markings allow each agent to find the currently known shortest path from the current position back to the starting point. This path is needed for the return phase of the algorithm. Additionally the agents are able to exchange data directly via wireless communication between themselves, as long as some restrictions are adhered to. These restrictions are a line of sight connection between the two agents and that they have to be in a specific range of each other. The result of these constraints is, that an error free wireless communication can be assumed between two communicating agents. This data transfer model allows for an exchange of maps, build by each agent based on the gathered data from the environment. The maps also contain the same step counters that the agent marks in the environment. After each map exchange the agent merges the newly acquired map with its own internal map (by using this algorithm for example [15]).

Should the direct communication between the agents not be possible, the algorithm is still able to finish its execution, as the agents can use the markings dropped in the environment, which are part of the indirect communication model. The combination of the two different communication types allows for a graceful degradation of the direct data transfer and enables the algorithm to continue to work in the worst case, c.f. [16], [10]. Another result of the re-introduction of wireless communication for the MAF algorithm is the possibility to create one or more ad-hoc networks between the various agents. Using these networks will allow the agents to share the collected map data between all participants of the specific network, allowing for an upsurge of data exchange and the faster propagation of the assembled data.

As soon as an agent finds a point of interest, or in this case a victim, the agent will switch from the search mode into the return mode. Now the agent tries to return to the starting point, respectively the head quarter of the search & rescue mission, to share the information about this found victim and to enable a subsequent rescue of the victim by the human team members. The agent now uses the stored map data to select the currently shortest known path back to the base. Additionally it also checks the markings that it encounters on its way back and compares the data stored in the markings with the data stored in its map. This in turn allows the agent to update the map data and the agent can now also react to newer information stored by the other exploring agents in the marking, as the map may not always be up to date if the agent has not encountered other agents for some time. As soon as the agent has reached the head quarter it resets its distance counter and it switches back into the search mode to start a new search for other possible victims in the terrain.

B. Frontier-based exploration

As described above the agents of the MAF algorithm decide randomly which part of the terrain should be explored, only depending on the markings on the ground to distinguish between already visited and yet uncovered terrain. With the introduction of direct communication to the MAF algorithm in [10] the agents are also able to create and store a map. This map can be exchanged between the agents and each agent is able to merge the received maps with its own map. This map can now be used to not only aid the agents to return

to the starting point but it can also be used to speed up the exploration process. By dividing the data stored in the map into two different parts, explored and unexplored terrain, the agent is now able to specify points which will offer the most "new information" to the agent by exploring this specific point. The division between these two different parts of the map is called a frontier. The calculated points chosen by an agent are called in turn frontier points. If an agent now chooses a frontier point, moves towards this point and updates its map based on the collected data, then the agent is able to calculate new frontier points based on the newly acquired data. By repeating this action, choosing a point, updating the map, and choosing a new point the agent will explore the unknown terrain faster, than by simply moving around randomly. This way of exploring the terrain is called frontier-based exploration as proposed by Yamauchi in [17]. This approach is also viable for the use in multi-agent algorithms, as the agents will choose the points based on their individual data. The additional data exchange based on the interaction between the agents will enhance the exploration rate as some parts may already be explored by another agent, compare [18]. The only drawback of this approach is the amount of calculations that have to be done. Especially if the maps are bigger.

To overcome this drawback Keidar and Kaminka introduced the Wavefront Frontier Detection algorithm (WFD) in [19]. The terrain on the map has only to be split into two parts: explored and unexplored terrain. Instead of iterating over the whole map, the algorithm just has to iterate over these regions. This in turn decreases the calculation time of the algorithm, notably if most parts of the map are still unexplored.

C. Pathfinding algorithms

The problem of finding a path between two different points is well known and various algorithms exist to solve this problem. For example the A* algorithm, based on Dijkstra's algorithm, is a static algorithm that will calculate the shortest path between two points, [12]. The main drawback of this algorithm is that it is a static algorithm and changes in the environment will force a recalculation of the whole path. The same problem applies to the Jump Point Search algorithm, which is based on the A* algorithm, [13]. Although it offers a faster calculation as the A* algorithm, a change in the environment will also force a recalculation of the path.

Another option is the *Bug2* algorithm devised by Lumelsky, [14]. This algorithm may not offer the shortest path between two points, but it is able to react to dynamic changes in the environment without starting a recalculation and the whole computation is rather light weight. The concept of the algorithm is rather simple: the agent will try to travel to the target point on a straight line until an obstacle is encountered, which the agent will now try to circle around to continue its way towards the target point.

Fig. 2 shows an example of a path of an agent, see also [20, p. 13]. The agent starts at point S and tries to reach point T. As soon as the agent encounters the first obstacle, it will circle around it clockwise. The algorithm will calculate two

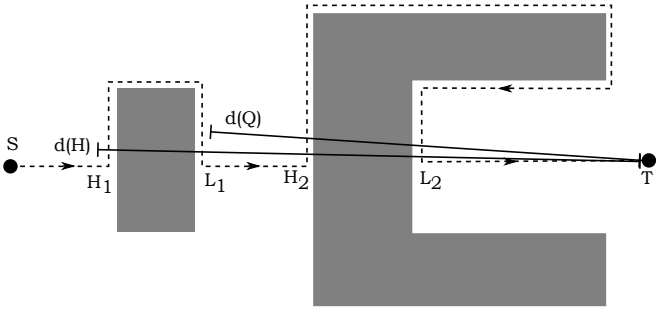

Fig. 2. Example of the Bug2 algorithm

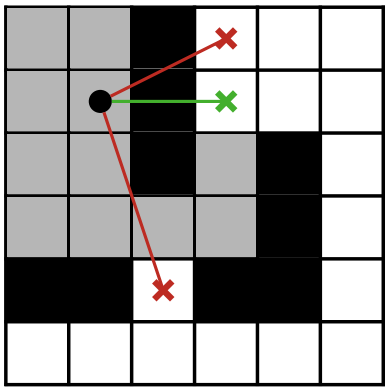

Fig. 3. List of available frontier points, with the chosen point in green and discarded points in red

distances, $d(H_i)$ and $d(Q)$. $d(H_i)$ is the distance from the entry point of obstacle i and the point T. $d(Q)$ is the distance from a possible leave point to the point T. If $d(Q) < d(H_i)$, then the leave point for obstacle i is $L_i = Q$, as long as the line QT does not cross i at the point Q.

III. Extending the MAF Algorithm with Frontier-Based Exploration

The Multi-Agent Flood algorithm does not have to be modified all that much to allow the frontier-based exploration to work. The agents are already able to store a map and they are able to exchange the information between themselves as soon as they are in range to allow a wireless data transfer. By including the Wavefront Frontier Detection algorithm in the MAF algorithm the agents are now able to calculate a list of possible frontier points to speed up the exploration. The MAF agent will now select a point based on two criteria: the euclidean distance from the point to the agent and the size of unexplored terrain around the specific frontier point.

An example of the possible frontier points is shown in Fig. 3. The agent, represented by the black dot, has already explored some terrain, shown in light gray. The unexplored terrain is depicted in white and obstacles in black. The green and red points show the possible frontier points, whereas the green point is the point that fulfills the two criteria: it is the nearest point and it has the most unexplored terrain around its position.

The problem that arises now is that although the point matches the criteria, the time it will take the agent to move towards this point may be longer, than the travel time to another point, for example the point in the bottom of the picture. To oppose this behavior a time limit was introduced. The agent now has to reach the point before the time limit is up or the point will get marked as unreachable and the robot will recalculate the frontier points to chose a new one, [18], [21]. Points marked as unreachable will be ignored.

After including the WFD algorithm in the Multi-Agent Flood algorithm the agents can now use the *Bug2* algorithm to reach the different frontier points, while circling around obstacles. The time limit introduced in the previous paragraph also enables the agents to dismiss frontier points that may not be reachable or too far away, as other obstacles may block the way.

The last point that has to be addressed is the cooperation between the agents. As of now the agents are able to calculate the frontier points and move towards these points. But as each agent is choosing these points independently from the other agents, it will happen that multiple agents will try to explore the same frontier point, resulting in a redundant exploration. To counter this excessive exploration the agents will now also exchange their current frontier point as soon as they initiate a map exchange. This allows the agents to check if the targeted point was already explored. If this is the case the agents will pick a new frontier point. If both agents try to explore a point that is still unexplored, the agents will calculate their distance from their current position to the targeted frontier point. The agent with the lower distance will keep its target and the other agent will pick a new point from its list of points. This algorithm is shown in Listing 1.

Algorithm 1 Coordination of the frontier points between the agents

```
if otherAgent.isInRange() then
  exchangeMap(otherAgent);
  if goal is explored then
    frontierpoints = WFD();
    goal = extractTargetPoint(frontierpoints);
  end if
  while goal is in range of otherAgent.getGoal() and
  otherAgent.getDistance() < this.getDistance() do
    ignoreList.add(goal);
    goal = extractTargetPoint(frontierpoints);
  end while
end if
```

IV. Experimental Simulation Setup

Five different test cases were used for the experimental simulation. One map with the size of 300×300 cells, three maps with a size of 500×500 cells, and a bigger scenario with a size of 1000×1000 cells. The maps used as test cases are representing often found environments in urban search and rescue scenarios, for example a floor plan of a house or the outline of a park. The the bigger map contained 15 points of interest uniformly distributed through the terrain, while the medium sized maps and the smaller map contained ten points of interest also uniformly distributed in the traversable space.

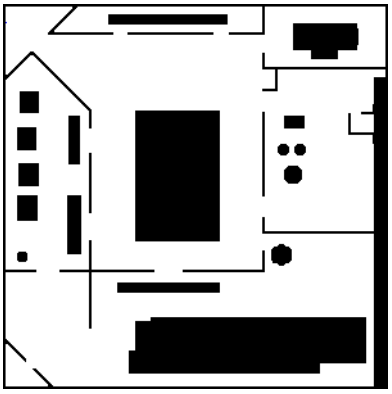

Fig. 4. Test case Factory, 300×300 cells

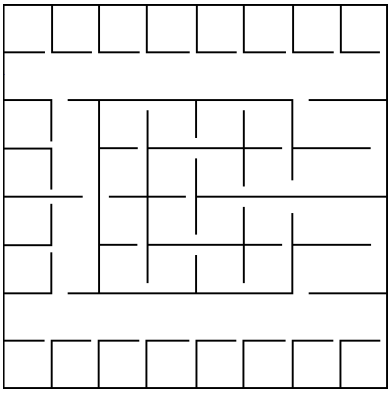

Fig. 5. Test case Cubicles, 1000×1000 cells

Fig. 4 and Fig. 5 show two examples of the test cases used. Three different algorithms were run, the original Multi-Agent Flood algorithm (as introduced in [11]) and two variants of the adapted MAF algorithm combining the frontier-based exploration, the path finding, and the increased cooperation. Each one was simulated using thirty agents (see also [22]). The two variants differ in only a small part: the first one, named *Bug*, is as described above, the second one, titles *BugDistributed*, has a random set of frontier points specified from the start. This difference should offer a faster spread in the first iterations of the simulation. Otherwise the behavior of the two versions is the same.

Each simulation was run until either 95% of the terrain was explored and 95% of points of interest were found, or the time limit was reached. The time limit depends on the size of the test case: the limit was the product of the height and width of each test case. The time is measured in steps in the simulation, whereas one step is the time it took all agents to move at least one cell. This in turn means that the algorithms had 90.000 steps to finish the small map, 250.000 steps to finish the medium maps, and 1.000.000 steps to terminate on

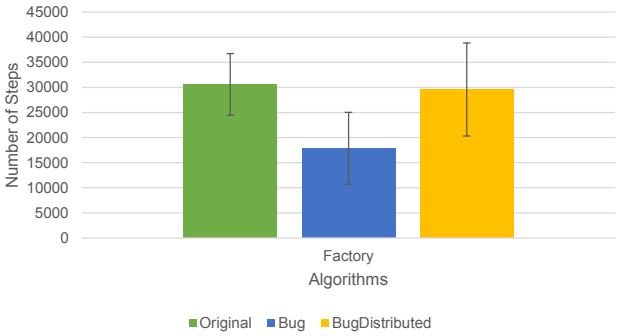

Fig. 6. Results, Factory, 300×300 cells

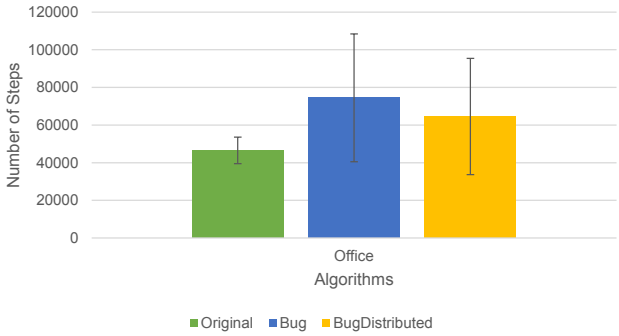

Fig. 8. Results, Office, 500×500 cells

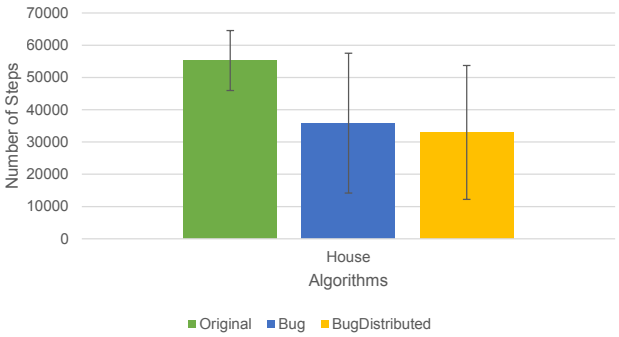

Fig. 7. Results, House, 500×500 cells

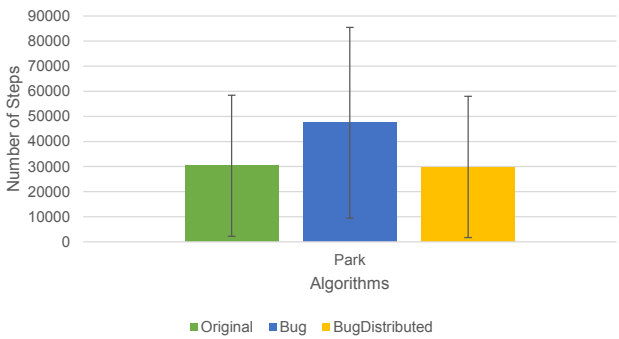

Fig. 9. Results, Park, 500×500 cells

the bigger map successfully. After the limit was reached the simulation was terminated unsuccessfully.

Each simulation configuration was repeated thirty times to calculate an average result.

The framework used for the implementation of the algorithm is the open-source multi-agent library MASON, written in Java, c.f. [23], [24], [25].

V. Simulation Results

The following bar charts depicted in Fig. 6 to 10 show the averaged results of all the simulation runs, they include also the runs that did not terminate successfully.

The three bars represent the three different algorithms. The green bar depicts the results of the original algorithm, the blue bar shows the results of the *Bug* algorithm, and the yellow bar shows the result of the *BugDistributed* algorithm. The Y-axis illustrates the number of steps. The T-capped lines centered on each bar depict the 99% confidence interval for the thirty repeated simulation runs.

As expected the merge of the Wavefront Frontier Detection algorithm and the Bug2 algorithm into the Multi-Agent Flood algorithm will speed up the run time of the MAF algorithm. This can be seen in the outcome of the simulation from the Factory test case, Fig. 6. The additional spread of the agents leads the *Bug* algorithm to only need about the half of the time it took the original algorithm to finish the simulation. The steps that the worst iteration of the *Bug* simulation needed to finish successfully was about the same steps it took the best iteration of the original algorithm. The *BugDistributed* variant is slighter faster than the original, but the variance is also a little bit higher.

On the other hand the results from the House test case, represented in Fig. 7, show a different outcome. The two newer variants of the algorithm offer a shorter mean run time than the original algorithm. Nearly needing only half the time than the original version. Although as it was the case in Fig. 6 the variance is greater in the *Bug* and *BugDistributed* than in the original algorithm.

The other results from the medium sized test case deviate from the first observations, shown in Fig. 8 and Fig. 9. The original algorithm is faster than the newer versions. The *Bug* variant is the slowest one in these two cases and the *BugDistributed* has either the same run time, as in the Park scenario, or takes roughly 50% more time than the original algorithm in the Office scenario. On the other hand, especially in the Office scenario, the configurations that needed the least amount of steps are nearly the same. The *Bug* and *BugDistributed* algorithms were a little bit faster (20815 and 21765 steps respectively, compared to 23974 steps needed by the original algorithm).

The results for the two newer variants are even worse in the bigger scenario, as depicted in Fig. 10. Out of thirty

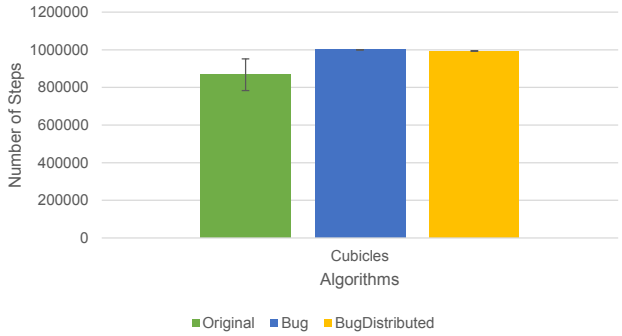

Fig. 10. Results, Cubicles, 1000×1000 cells

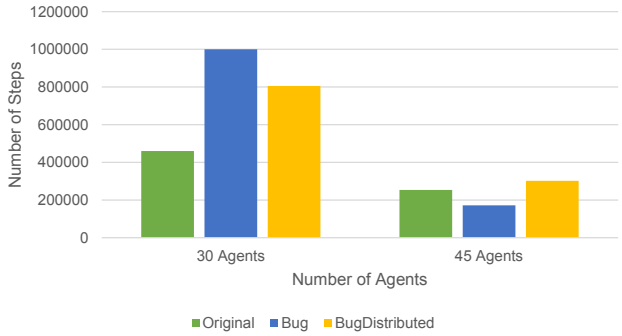

Fig. 11. Comparison between 30 and 45 Agents, Cubicles, 1000×1000 cells

repetitions of the simulation, fifteen simulations of the original algorithm finished in time and terminated successfully. Only one simulation run of the *BugDistributed* variant was able to finish in time, whereas none of the *Bug* variant could beat the time limit.

These results imply, that the number of agents the adapted Multi-Agent Flood algorithm needs to terminate successfully on the bigger maps needs to be increased. A new configuration of simulations was created and repeated thirty times. This time instead of using 30 agents the algorithm had 45 agents available. The new configuration was tested on the bigger 1000×1000 map, presented in Fig. 5.

The results of this simulation run are shown in the chart shown in Fig. 11. The green bar shows the minimum result of thirty simulations using the *Original* algorithm, the blue bar depicts the result of the *Bug* algorithm, and the yellow bar illustrates the outcome of the *BugDistributed* algorithm. The bars on the left side of the diagram depicts the lowest number of steps that the specific algorithm needed to finish the simulation successfully using 30 agents. The right side of the chart presents the lowest number of steps that the algorithm needed while using 45 agents. The use of 15 additional agents reduces the run time of the adapted algorithm to a quarter or a third of the run time of the algorithm using only 30 agents. Although the count of successful simulations is still quite low, only about one or two in thirty simulations finished successfully (corresponding to a percentage of about 3.3% - 6.6%). Using additional agents with the original algorithm yields also quite good results, as can be seen on the right side of Fig. 11, especially as the percentage of successful simulations is much higher than in the adapted variant (about 96.6%).

The outcome of the simulations suggests that the adapted Multi-Agent Flood algorithm is able to cut down the run time as long as there are enough agents available correlating to the map size. Due to the restrictions imposed on the direct communication part, the algorithm needs more agents as the spread implied by the adaption of the Wavefront Frontier Algorithm will work against the limited range which allows a direct wireless communication. Increasing the number of agents available in the same space will raise the density and in turn will raise the possibility of wireless communication, leading to an increased cooperation between the agents.

VI. Conclusion

In this paper we discussed how existing techniques can be used to decrease the run time of the Multi-Agent Flood algorithm. By adding the frontier-based exploration concept to the algorithm the agents are now able to discern specific points in the terrain which would lead to a better exploration of the unknown area. The next step was to introduce way finding algorithms, as the agents were now able to chose distinct frontier points. The choice of path finding algorithm fell onto the *Bug2* algorithm, as it offered a very computational light weight alternative to other algorithms. The last component added to the MAF algorithm was the ability of the agents to share and compare the frontier points. This in turn allowed them to decide whether an agent should explore the same point as an other agent. As the agents were now able to communicate their goals to other agents the spread of the agents through the terrain increased a lot. As a consequence the number of agents needed to utilize these adaptions fully was raised. The results have shown that with the proper number of agents corresponding to the size of the map the run time of the original algorithm could be cut down to a third or even a quarter.

Future work could take a look at how much the number of agents has to be increased to fully use the advantages offered by the direct communication part of the communication model again, especially in the larger maps. Otherwise different available path finding algorithms could be tested. Especially if the computational power does not matter all that much. This could decrease the run time even more, as the agents would now be able to reach the designated frontier points more effectively. Another point which could be more examined is the cooperation of the agents. Currently the agents try to keep out of the way of other agents, which in turn decreases the effectivity of the direct communication due to the imposed restrictions. By changing the way in which the agents decide to ignore points that other agents want to explore this decrease could be countered. Either by increasing the number of used agents or by changing the way the cooperation between the agents work to get them to maximise the utilization of the direct communication could also lead to an increased ad-hoc

network creation or better maintenance of an established ad-hoc network between the agents. This would boost the data transfer capability between the agents enormously.

REFERENCES

[1] A. Davids, "Urban search and rescue robots: from tragedy to technology," *IEEE Intelligent Systems*, vol. 17, no. 2, pp. 81–83, 2002.

[2] R. R. Murphy, S. Tadokoro, D. Nardi, A. Jacoff, P. Fiorini, H. Choset, and A. M. Erkmen, "Search and rescue robotics," *Handbook of Robotics. Springer*, pp. 1151–1173, 2008.

[3] R. R. Murphy, *Disaster robotics*. MIT press, 2014.

[4] K. Nagatani, S. Kiribayashi, Y. Okada, S. Tadokoro, T. Nishimura, T. Yoshida, E. Koyanagi, and Y. Hada, "Redesign of rescue mobile robot Quince," in *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE, 2011, pp. 13–18.

[5] S. Koenig and Y. Liu, "Terrain coverage with ant robots: a simulation study," in *Proceedings of the fifth international conference on Autonomous agents*. ACM, 2001, pp. 600–607.

[6] E. Ferranti, N. Trigoni, and M. Levene, "Brick & Mortar: an on-line multi-agent exploration algorithm," in *IEEE International Conference on Robotics and Automation*. IEEE, 2007, pp. 761–767.

[7] ——, "Rapid exploration of unknown areas through dynamic deployment of mobile and stationary sensor nodes," *Autonomous Agents and Multi-Agent Systems*, vol. 19, no. 2, pp. 210–243, 2009.

[8] J. De Hoog, S. Cameron, and A. Visser, "Selection of rendezvous points for multi-robot exploration in dynamic environments," in *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2010.

[9] M. Becker, F. Blatt, and H. Szczerbicka, "A multi-agent flooding algorithm for search and rescue operations in unknown terrain," in *Multiagent System Technologies*. Springer, 2013, pp. 19–28.

[10] F. Blatt, M. Becker, and H. Szczerbicka, "Optimizing the exploration efficiency of autonomous search and rescue agents using a concept of layered robust communication," in *IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*. IEEE, 2015, pp. 1–6.

[11] F. Blatt and H. Szczerbicka, "Realisation of navigation concepts for the multi-agent flood algorithm for search & rescue scenarios using rfid tags," in *IEEE/ACM 20th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. IEEE, 2016, pp. 112–115.

[12] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[13] D. D. Harabor, A. Grastien *et al.*, "Online graph pruning for pathfinding on grid maps." in *AAAI*, 2011.

[14] V. Lumelsky and A. Stepanov, "Dynamic path planning for a mobile automaton with limited information on the environment," *IEEE transactions on Automatic control*, vol. 31, no. 11, pp. 1058–1063, 1986.

[15] M. Pfingsthorn, B. Slamet, and A. Visser, "A scalable hybrid multi-robot SLAM method for highly detailed maps," in *RoboCup 2007: Robot Soccer World Cup XI*. Springer, 2008, pp. 457–464.

[16] M. Becker, F. Blatt, and H. Szczerbicka, "A concept of layered robust communication between robots in multi-agent search & rescue scenarios," in *IEEE/ACM 18th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. IEEE, 2014, pp. 175–180.

[17] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *IEEE International Symposium on Computational Intelligence in Robotics and Automation*. IEEE, 1997, pp. 146–151.

[18] ——, "Frontier-based exploration using multiple robots," in *Proceedings of the second international conference on Autonomous agents*. ACM, 1998, pp. 47–53.

[19] M. Keidar and G. A. Kaminka, "Robot exploration with fast frontier detection: theory and experiments," in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*. International Foundation for Autonomous Agents and Multiagent Systems, 2012, pp. 113–120.

[20] N. S. Rao, S. Kareti, W. Shi, and S. S. Iyengar, "Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms," Citeseer, Tech. Rep., 1993.

[21] A. Baranzadeh and A. V. Savkin, "A distributed control algorithm for area search by a multi-robot team," *Robotica*, pp. 1–21, 2016.

[22] F. Blatt, M. Becker, and H. Szczerbicka, "Analysing the cost-efficiency of the multi-agent flood algorithm in search and rescue scenarios," in *German Conference on Multiagent System Technologies*. Springer, 2016, pp. 147–154.

[23] S. Luke, G. C. Balan, L. Panait, C. Cioffi-Revilla, and S. Paus, "Mason: A java multi-agent simulation library," in *Proceedings of Agent 2003 Conference on Challenges in Social Simulation*, vol. 9, no. 9, 2003.

[24] S. Luke, C. Cioffi-Revilla, L. Panait, and K. Sullivan, "Mason: A new multi-agent simulation toolkit," in *Proceedings of the 2004 swarmfest workshop*, vol. 8, 2004, p. 44.

[25] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan, "Mason: A multiagent simulation environment," *Simulation*, vol. 81, no. 7, pp. 517–527, 2005.