algorithms

MDPI

Article

Balanced Parallel Exploration of Orthogonal Regions

Wyatt Clements [1,*], Costas Busch [1,*], Limeng Pu [1], Daniel Smith [2] and Hsiao-Chun Wu [1]

[1] School of Electrical Engineering and Computer Science, Louisiana State University, Baton Rouge, LA 70803, USA; lpu1@lsu.edu (L.P.); hwu1@lsu.edu (H.-C.W.)

[2] School of Biological and Agricultural Engineering, Louisiana State University, Baton Rouge, LA 70803, USA; dsmi112@lsu.edu

* Correspondence: wcleme3@lsu.edu (W.C.); kbusch@lsu.edu (C.B.)

Received: 28 January 2019; Accepted: 9 May 2019; Published: 15 May 2019

Abstract: We consider the use of multiple mobile agents to explore an unknown area. The area is orthogonal, such that all perimeter lines run both vertically and horizontally. The area may consist of unknown rectangular holes which are non-traversable internally. For the sake of analysis, we assume that the area is discretized into N points allowing the agents to move from one point to an adjacent one. Mobile agents communicate through face-to-face communication when in adjacent points. The objective of exploration is to develop an online algorithm that will explore the entire area while reducing the total work of all k agents, where the work is measured as the number of points traversed. We propose splitting the exploration into two alternating tasks, perimeter and room exploration. The agents all begin with the perimeter scan and when a room is found they transition to room scan after which they continue with perimeter scan until the next room is found and so on. Given the total traversable points N, our algorithm completes in total $O(N)$ work with each agent performing $O(N/k)$ work, namely the work is balanced. If the rooms are hole-free the exploration time is also asymptotically optimal, $O(N/k)$. To our knowledge, this is the first agent coordination algorithm that considers simultaneously work balancing and small exploration time.

Keywords: online algorithm; mobile agents; parallel exploration; limited communication; work balancing

1. Introduction

In mobile agent research, there are two types of problems: coverage and exploration. Coverage is concerned with patrolling an area, i.e., how to organize the agents so they can maximize the area they can view while still being able to be in contact range of the other agents in a chain [1–4]. This assumes that the agents already have full understanding of the area to be able to determine the best course of action to achieve these goals. On the other hand, exploration works with a previously unknown area, and attempts to map all the area. Applications of area exploration include to find survivors quickly, setup a network, or to rendezvous distributed agents. Hence, exploration is a more fundamental problem which may also be used to solve coverage problems.

To explore an unknown area, previous papers have used sensor networks [5–9]. This style of exploration has the agents placing static sensors at intervals of the area to create a network and cover the explored area. The sensors then can be used by the agent to find the area boundaries and store information from previous scans of the robots that can help to explore new parts of the area. The agent itself has little memory and relies heavily on the sensors. The sensors must be carried by the agents adding weight that inhibits the mobility.

In this paper, we present a set of algorithms to solve area exploration without using auxiliary static sensor nodes, but rather, with just self-reliant mobile agents. The area is an orthogonal closed region that is made up of horizontal and vertical walls. This area is then discretized into N traversable points that are connected to their neighboring points in the cardinal directions. An agent moves north, east,

south, or west, on the neighbor points from its current position, whenever these points are available. The number N is not known to the agents. The agents discover the area and the number of points during exploration. The area contains holes, non-traversable locations, which are randomly distributed in the area; for example, Figure 1 depicts black holes in the area. These holes are rectangular and have a length and width within an aspect ratio a. The agents begin in a chain along the perimeter of the area, Figure 1 black xs. To close area 1, Figure 1, at the time of reaching the first and second corner a room cannot be closed, it is only after reaching the third corner in this scenario to close a room. This is due to the agents not knowing the composition of the room causing them to have a previously visited x coordinate with the third corner.

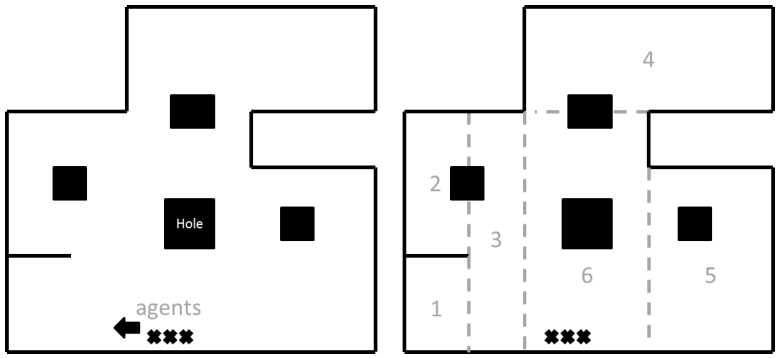

Figure 1. Example of Area and Generation of Rooms.

All agents are assumed to be the same with equal traversal ability, communication range, and total memory. An agent occupies a single point at a time and can move one point in the cardinal directions in which the point has connection. We assume that it takes one time step to move point to point. The agents have enough memory to map the entire area (i.e., find coordinates of explored points) which can be shared between agents. To share area information, the agents must be within adjacent points, or in a connected chain in which there is a sequence of adjacent pairs of agents. Agents share information on their total area traversed as well as the orientation of the area. Through this process, agents can make decisions, such as room scan roles, and which one will be the front agent for perimeter scan. These constraints on the agents enables them to not rely on sensors to have knowledge of the unexplored area. The work of an agent is the total number of points it traverses. Each agent keeps track of the work it has performed, and the goal is to maintain balanced work among the agents.

The main goal is to explore an unknown orthogonal region of N points with k mobile agents in parallel in order to reduce the work, which is the total number of points visited, of each agent to explore the area to $O(N/k)$, namely balancing the work and total exploration time $O(N/k)$. The difficulty with this is the introduction of holes in the area. If the holes were not there, we could simply traverse the entire perimeter, divide the area into k equal partitions, and send the agents to their respective area. This is not possible with holes, as one or more of these partitions could consist mainly of holes, so the agents associated with those partitions would not perform $O(N/k)$ leading to an imbalance of work between all agents. Thus, we developed an algorithm that handles the unknown holes in the area to maintain balanced work.

The difficulty with these holes is that they are arbitrarily placed in the area. As a sub area was closed, if we simply divided the area into k partitions, then we cannot guarantee that all the agents perform near equal work as they can only communicate within a single grid point. For example, let's assume that the hole in a room of n points takes up all space except a single path around the area, then if partitioned, each agent encounters the same hole resulting in repeated work, which involves traversing around the hole and performing nk total work in the area instead of n total work. Moreover, as another example one agent may have to explore smaller number of holes (or total points) than other agents again resulting in imbalanced work.

To get the near equal work for each agent, we have introduced the concept of Leader, Active, and Inactive agent roles when an area is being closed (room created) and explored. These roles are determined based on the amount of work that was previously performed in other closed areas, the most, Leader, the least, Active, and all the rest, Inactive. Leader organizes the other agents to maintain their overall work when Active return to the Leader. Active performs traversal and when enough work has been completed, Active returns to the Leader to maintain equal work. Inactive remain with the Leader until an Active returns sharing its work making them Active. When the closed area is completed the agents return to being against the perimeter to traverse to the next closed area. When a room is discovered then all the agents move to explore the room. Inside the room, the exploration is performed in parallel by the agents while balancing their work. The reason of this approach, of exploring the rooms one after the other in sequence, is to have an online exploration of the whole area in one pass of the perimeter. Otherwise, we would first explore the whole perimeter, and then split the area into rooms, but this would have required to revisit the perimeter. Such a process would also not give us balanced work as large chunks of the area that most of the agents are sent might be made up of holes, unbalancing the agents' work.

In our algorithm, the agents start along all adjacent to each other next to the perimeter. Then they traverse the perimeter maintaining a convoy. During the traversal, the agents create rooms to explore and map. The whole area is divided on the fly into a sequence of rooms, which are explored one after the other.

In our approach, each room is a rectangular closed area which is identified on the fly at corners of the perimeter. When a room is identified the agents move immediately into the room together and explore all the points of the room by dividing the work among each other. A room may contain obstacles which are not immediately detected, and hence, the area of the room may not be able to be divided equally among the robots, which may cause imbalance of the work. To alleviate this problem, the agents start the room exploration slowly, and use exponential delay and meeting times when exploring the room to handle the unknown holes in the room. Agents that exceed their allocated work quota give future work to other agents to maintain parallelism and load balancing.

Paper Outline

The paper will continue as the following: Section 2 has the related work, Section 3 introduces the problem and agent description, Section 4 our algorithms, Section 5 the analysis with respective results, and Section 6 gives conclusion and future work.

2. Related Work

The most related previous work is solving a similar problem, using "oriented disjoint rectangular obstacles" in an $n \times n$ grid [10]. In this, the agents also use only local information, but with the oriented obstacles, all obstacles follow an oriented pattern that allow the area to be divided into similarly shaped partitions. Our problem has arbitrarily located rectangular obstacles of ratio a. With this, our problem or theirs cannot be reduced to the same problem. For this reason, a new algorithm had to be developed not only to traverse the entire area, as it does not follow their $n \times n$ grid rules with disjointed obstacles, and another algorithm for the room, as the obstacles do not follow the same constraints. Finally, our focus is on balanced work as well as parallel while theirs is only on parallel work.

The most closely related topics in the literature are "map merging", concerned with multiple-robot simultaneous localization and mapping (SLAM) [11], and "map synchronization" [12], where the agents share the map information of the area explored. Agents are not assumed to be together and use probabilistic generalized Voronoi diagram to, "achieve fast and accurate map fusion for large maps"[11]. In map synchronization, agents can form into subgroups that are close enough to have the same map, while other agents that are not will synchronize maps when they approach [12]. This is also similar to Frontier based, which is concerned with exploring all points of a grid area in a minimum

amount of time [13,14]. However, these do not consider load balancing the work of the agents as we do here.

One topic of focus on this area is agents that can fail. This can be through "black hole" or "black link" in an edge or link in a sensor network [15], or the agent itself can fail [16]. To be able to explore the entire area, the number of sensors must be more than the number of edges plus two times the number of links [15]. With using "pheromones", Levy Walk was determined to outperform random walk when exploring, especially when failure prone agents are introduced [16].

Exploration is also a topic of interest in communication networks. Ad-Hoc networks are constantly changing, mobile agent routing leaves information at nodes it visits and reads the information that is left at a node [17,18]. In this way, the agents can cooperate much like ants leaving information to share with others when looking for food. When speed of network changes increases the delay decreases but roughly only 68% of packets are received and this amount drops greatly as speed increases [17]. MAR [18] can perform at greater speeds with the same percentage of packets received but less delay.

Machine learning algorithms have also been used for mobile agent exploration. Q-Learning has been used as a "Human-like" learning algorithm to manage energy, avoid obstacles, working towards goals, and following a line [8]. Another implementation used online-learning Bayesian network to predict terrain range and magnetic field intensity [19]. A second online-learning method used Gaussian Processes [20]. In this learning, decentralized agents, flying quadcopters, are used to explore an area by mapping the distance of an unknown environment [20].

Agents have also been given roles or working with a stationary agent to explore an area. For search and rescue, agents are given a larger environment than what they can explore while staying in range so they must go out of communication range with agents being declared explorers or relays [21]. Area is shared from the explorer to the relay and then back to the command center [21]. Another case is when the agent must communicate with a computer when an obstacle is located so that the computer can determine the path to take [22]. The agents can also make a decision given multiple criteria to locate and rescue victims within a limited amount of time [23]. This MCDM model, used in search and rescue does not attempt to explore the whole map, but to use the knowledge of the agents exploration to move quickly to locations that the victims are believed to be, in this manner it finds the majority in the least amount of time [23].

Similar to search and rescue, research has also been conducted for agents to rendezvous before exploration of the area [24]. In this research, agents look for "landmarks" to attempt to see if any other agents are within range of their sight or if they can find any other "landmarks" to search from there [24]. This research does not focus on the exploration itself, but attempts to minimize the time for agents to rendezvous.

Constraints have been set on the power of agents, they are assumed to have limited power and must work together to find "targets" distributed in an unexplored area. The first strategy is to use a similar technique as ants, drop pheromones, to tell other agents this area has been explored, or there is a target in this direction [25]. This strategy does not require communication, so it is able to save more energy for exploration. Next, the technique is to use communication between agents similar to how fireflies communicate [25]. The downside to this strategy is that it consumes energy to communicate. The more complex the task the better firefly approach performs compared to the ant approach.

3. Model

The problem we are exploring consider an orthogonal area on the plane. Through this, the perimeter edges all meet at right angles. There can also be interior lines within the interior which are extension of the edges from the perimeter. It is assumed that the whole orthogonal area can be discretized into grid points that make up the entire orthogonal area. We assume that grid points are spaced one unit distance apart. Each point is adjacent to those (up to four) neighbor grid points that are one unit distance in the cardinal directions (north, east, south, west).

We assume that all agents have the same sense of common cardinal directions, where they all know where is north, east, south, and west. This helps them to agree on a common clockwise direction for traversing the perimeter. An agent is dimensionless and resides in exactly one grid point at a time. Two agents directly communicate (they are in communication range with each other) if they reside in adjacent grid points (one unit distance away). However, they can also indirectly communicate with each other through a chain of agents such that each pair along the chain reside in adjacent grid points. The agent visibility range includes the four adjacent points in the cardinal direction. Agents can maneuver on the grid points moving from the current grid point to an adjacent grid point.

In the area there are rectangular holes such that no grid point exists in the hole making them non-traversable. These holes are unknown until discovered by an agent. Each hole in the area has an aspect ratio a such that a hole can be of any size as long as the width and length are bounded by the aspect ratio; that is, the width over length ratio is bounded by a, and length over width ratio is also bounded by a. Each hole must have at least two unit distance separating it from another hole or from any edge of the perimeter (or internal edge). In this way, at least one grid point fits between the holes or between a hole and an edge. To completely discover a hole, an agent must traverse around the hole, which is the only way to detect whether it is a hole or part of the perimeter.

A unit of work corresponds to visiting a grid point. To completely explore the area, all the grid points that make up the area must be visited, and the information is shared by each agent. The agents have enough memory to map the entire area as well as recording the amount of work performed by all agents, both perimeter and room work. A worst case bound for the total amount of memory that an agent would need to have to map the entire area would be storing N coordinates, corresponding to the total number of grid points in the area. In addition, an agent may store information to track the amount of work performed by the other agents. Agents share the information of what they have encountered, including holes and grid points, when they return to communication range. All agents are assumed to be identical; able to traverse a single grid point at each time step, communication range of one grid point, vision of one grid point to map the area surrounding the agent. Each grid point that an agent moves counts as one unit of work. Multiple agents can all communicate with one another through a chain of agents.

During the perimeter work, agents move together maintaining a chain along the perimeter. Agents move together to find a room and then perform the room scan algorithm in parallel. As this area is unknown the agents must follow in this convoy as communication is limited. In an attempt to close a room, the front agent will attempt to leave the perimeter to reach a previously explored grid point. If able to close, then the front agent returns to the other agents and perform the room work, otherwise, the front agent returns, and they continue traversing the perimeter. After completing room work, the agents determine, through their chain of communication, which agent has the least total perimeter work and that agent is moved to the front of the agents to lead the perimeter traversal.

When a room is encountered, then the agents must determine their role, Leader, Active, or Inactive. The one that is Leader is the one that has the most work, Active is the one that has the least work, and Inactive are all those in between. As agents communicate their total work, if two agents have most total work then we assume they have unique identifiers and the one with the lower identifier is made Leader. Leader communicates with the Active as they return to share or give their work to an Inactive to maintain equal room work. In keeping balanced work agents must return to the Leader for communication range to determine what to do with their work. Please note that even through the Leader is stationary, it already performed most of the work and the other nodes can perform the work, and in any case the Leader will be swapped by one the remaining $k-1$ agents that returns with more work. Active traverse the area, after a set amount of work if there remains work in the room, then the Active returns to Leader to determine what is to be done next. Finally, Inactive wait with the Leader to be given work that remains from the Active. Active agents know the location of the Leader and Inactive agents to return to communicate with the Leader. When the Active gains more work than the

Leader, the Active and Leader will change roles. Once the work is completed all agents will return to the Leader to organize into a line along the perimeter.

This process of performing perimeter work until a room is discovered and then performing room work is repeated until all points are traversed.

4. Algorithms

As mentioned in the introduction, the exploration algorithms can be separated into two parts, perimeter exploration and room exploration. In perimeter exploration the area is divided into rooms. Agents maintain a track of the work they perform during perimeter and the work they perform during room exploration separately. During perimeter work, agents will rotate the agent with the lowest perimeter work after a room is attempted to be closed to maintain balanced perimeter work. With room work, the agent with the most work will be in charge of coordinating the room work between other agents and transferring the responsibility to a new agent with more work whenever necessary.

4.1. Perimeter Exploration

The first algorithm performs the exploration of perimeter P (Algorithm 1) . When the exploration begins, the agents all traverse clockwise along the perimeter, depicted in Figure 1 by the black arrow. The agents follow the front agent in that direction maintaining a connected convoy (chain of agents), as discussed in Section 3, along the perimeter of the area, adding explored points to P. If the front agent is no longer against the perimeter or they have reached a perimeter wall along the path they notify the other agents that a corner has been reached at point $\delta = (X_t, Y_t)$. At this point, before continuing along the perimeter moving along the path, if a corner is encountered then the front agent looks at its mapping of the area to determine if they have already visited a perimeter point across, namely δ' such that $\delta' = (X, Y)$ where $X = X_t$ or $Y = Y_t$ and (X, Y) is in the visited P. The agent can find possibly up to three candidate δ' depending on exploration history of P, as shown for example in Figures 2a,b. Figure 2a,b depicts zooming in on a corner point such that in Figure 2a, the agent encounters the corner from traversing from the top, while in Figure 2b, the agent encounters the corner traversing from the left. Each of these candidate closures can be placed in a queue, without loss of generality, we can assume they are placed in the queue from cardinal direction north then clockwise around the front agent.

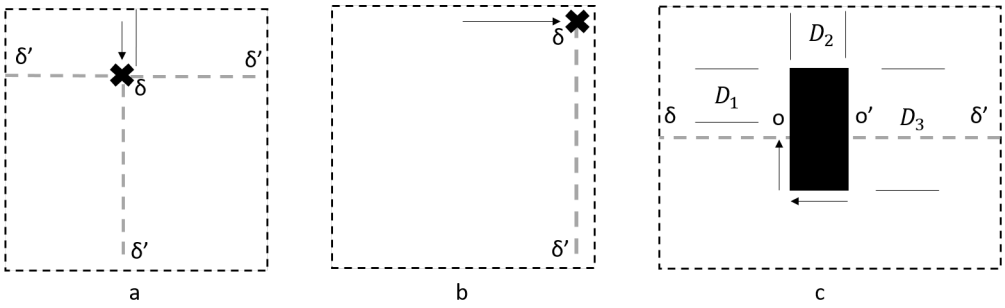

Figure 2. Possible room closure directions. (**a**) Maximum possible directions given internal wall; (**b**) Minimum possible directions given 90 degree perimeter angle; (**c**) Traversing around holes.

The front agent then takes the first direction in the queue and begins moving towards the picked δ' to determine if a room closure is possible. The front agent continues along a straight path until reaching the previously explored position δ', as for example depicted in room 1 generated in Figure 1 where no obstacle is encountered. Otherwise, as depicted in Figure 1 in the generation of room 3, if the agent detects an obstacle along its path then it must determine if the obstacle is a hole or a wall, which is determined by calling the obstacle subroutine Algorithm 2.

The front agent returns to δ if an unexplored wall was hit or the agent successfully reached δ'. After returning, if δ' was reached a room has been found, and then room scan is called, and the corner is marked as used. After performing room scan, or hitting a wall and returning, the front agent of the convoy switching with the agent that has the least amount of perimeter work completed, to balance the work in the perimeter. After all the δ' have been explored for that δ, the agents perform the turn and continue scanning the remainder of the perimeter.

Algorithm 1: PerimeterScan

```
/* Algorithm for agents to scan around the perimeter                    */
Each agent tracks its work done during perimeter scan;
All agents begin together and parallel to exterior wall;
Explored perimeter points P ← ∅;
while A perimeter point δ is left to be scanned (i.e., p ∉ P) do
    Agents move together as a convoy keeping a line adjacent to the exterior wall they are against;
    The front of the convoy will be on top of δ; P ← δ ∪ P;
    if convoy front agent performs a turn at δ = (X_t, Y_t) or there is an unused corner perpendicular to traversal direction then
        // Check possible directions for closing room
        RoomPoints ← the set of closest perimeter points (X, Y) ∈ P such that either X = X_t or Y = Y_t;
        Remove from RoomPoints all the points (X, Y) such that the straight line from (X_t, Y_t) to (X, Y) goes through
          a previously visited hole or wall that is not in P;
        // Attempt to close a room
        for δ' ∈ RoomPoints do
            WallHit, Sucess ← false;
            ℓ ← convoy front agent;
            while not (WallHit or Success) do
                Agent ℓ moves one point closer towards point δ';
                if agent ℓ has reached δ' then
                    Success ← true;
                end
                if obstacle encountered then
                    WallHit ← ObstacleCheck(δ, δ');
                end
            end
            Agent ℓ returns to δ;
            if Success then
                RoomScan(δ');
                mark corner as used;
            end
            Change front agent of convoy to that one with smallest perimeter scan traversal;
        end
    end
end
```

The obstacle subroutine Algorithm 2 determines if the obstacle encountered is a hole or a wall. Let the point that the obstacle is encountered be o. The agent checks that the first length of the obstacle D_1 (see Figure 2c), perpendicular to the line l between δ and δ', is shorter than the aspect ratio a multiplied by the length of l which is $Dist = |X_t - X| + |Y_t - Y|$. If this value is exceeded, then the obstacle cannot be a hole that fits between δ and δ'. Continuing from this point, the agent checks each dimension to make sure that the obstacle is a hole. It checks if both of the lengths D_2 and D_3, Figure 2c, are within the a aspect ratio, first between D_1 and D_2 and finally between D_2 and D_3. During the exploration to D_3 the agent will pass o' such that o' is on the line from δ to δ'. After this point, it is known that the obstacle adheres the aspect ratio restrictions of a hole, but the agent must make sure that it is rectangular by returning to o around the points that are between D_1 and D_3 that have not been explored. If it is indeed a hole, the agent adds in its memory as a known hole, a hole that is detected in the area, and returns to o'.

Algorithm 2: ObstacleCheck(δ, δ')

```
/* Algorithm to check if perimeter wall hit when closing a room */
Dist ← |X_t − X| + |Y_t − Y|, where δ = (X_t, Y_t) and δ' = (X, Y);
o ← point obstacle encountered;
Move to corner of obstacle (wall or hole), in the direction point of previous than δ in list P;
D_1 ← points traversed;
if D_1 > a · Dist then
    return true;
end
Turn along obstacle;
Move to next corner of obstacle;
D_2 ← points traversed;
if D_1 > a · D_2 then
    return true;
end
Turn along obstacle;
Move along obstacle until a · (D_2 + 1) points have been traversed or corner is reached;
D_3 ← points traversed;
o' ← point along δ and δ' found during traversal of D_3;
if D_3 < a · D_2 or D_2 < a · D_3 then
    Attempt to return to o along points not yet traversed;
    if obstacle is rectangular then
        Move back to o';
        return false;
    end
end
return true
```

4.2. Room Exploration

After front agent in perimeter scan successfully reaches δ' and returning to the other agents, a room is found, the room scan is invoked. Now that the dimension of the room is known, the agents could have divided the room into k equal areas and begin to search. However, given that the holes are arbitrarily placed, it can be such that the assignments of some agents may contain no work. The other agents cannot detect this until being within communication range. With this, it can also be such that the agents repeat the same work when traversing around a hole in the area. Therefore, to maintain balanced work, we introduce agent roles and a slow start to increase parallelism.

The agents are set to the roles of Leader, Active, or Inactive. All agents are within communication range at this point and communicate their work with the other agents to determine who has the most work, Leader, and the one with least work, Active, and the remaining, Inactive. If two share work then that agent with the lower ID will be Leader. The purpose of the Leader is to maintain a ratio of work between all agents to balance the work performed by each agent. The Active agent is given a task in the form of a stack of areas to explore, and performs the exploration until a work threshold is reached. This stack is initially a single task consisting of the entire room. As the Active agent performs the traversal it will split the current task and push it back on the stack. A task is two corner points of the perimeter of the area they are required to explore. In giving the Active agent these two points, the agent can discern the rectangle of exploration.

The Inactive agents are in a convoy by the Leader waiting for the Leader to give them a task and become Active. The allocation of the work to the agents is performed in a balanced way, such that those agents with lower work are given work and those that exceed the work of lower working agents are made Inactive. If this is the first room attempted to close then no agent has any room work so it is arbitrary which agent is set to Leader, Active, or Inactive. Otherwise, the agent with the most room work is set to the Leader, yellow 'x' in Figure 3, the one with the least room work is set to Active, black 'x' in Figure 3, and the rest begin as Inactive, white 'x' in Figure 3.

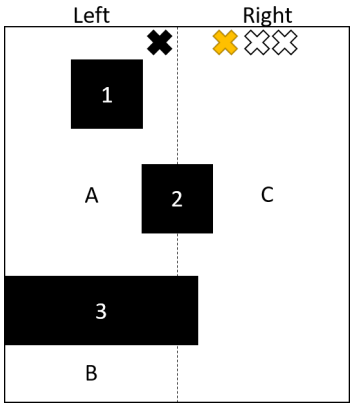

Figure 3. Splitting a Room with Holes.

The work threshold of an Active agent is initially set to a value d_{max} that is determined by the longest length of the room and the cost of traversing around holes on a straight path, discussed in more detail in Section 5 where we analyze the algorithm. As the room exploration progresses, the work threshold exponentially increases to accommodate the expected work in the newly discovered areas. The amount of work threshold only increases when work is not shared or given to another agent. This is due to the fact that if no other agent was there, then doubling the amount of work to return will maintain equal work among the agents. Gradually, Inactive agents are becoming Active as more area is discovered in the room.

The Leader subroutine (Algorithm 3) describes the Leader's role as Active agents return to the Leader. The Leader's job is to maintain the remaining work between itself and the Active and Inactive agents. When an agent i has performed work that exceeds its threshold or it ran out of tasks to perform, it returns to communication range of the Leader. In performing this the agents repeats the same amount of work as the threshold to return, but with exponentially increasing work threshold the agent will perform more new work than the cost to return to the Leader. This way, load balancing can be achieved at a faster pace. It is necessary to return to communication with the Leader to enable the Leader to balance the work between the Inactive agents. If the agent i has a stack remaining, or the Leader has a stack, then the Leader takes agent is stack and merges it with its own. The Leader must first make sure that the total work performed by agent i has not exceeded a ratio 3 between the Leader and agent i, if agent i did exceed this ratio, then the Leader and agent swap roles and the Leader begins the Active part of RoomScan. If at least one other Inactive agent j is waiting with agent i then the Leader must check the ratio of work between agent i and all Inactive agents j to decide what to do with the stack. If agent is work is within a ratio 4 of all other agents j then the Leader splits the stack in half, giving one part back to agent i and the other to the agent with lowest total work. After this, the Leader resets the threshold back to the minimum for agent i and signals agents i and j to become Active. Otherwise, if Inactive agent(s) j are waiting and i has exceeded the ratio, then the entire stack is given to the agent with lowest total work. Finally, if no other agent is Inactive and agent i is within a ratio of 4 between all the Active agents, then the stack is returned to agent i and agent i is signaled to remain Active.

The Active subroutine (Algorithm 4) describes the Active agents' role as they are given a task by the Leader. The agent i tracks their current work, and their current threshold of work to determine when to return to the Leader. While the agent has task(s) to accomplish and the new work they have completed has not exceeded the current work threshold, the agent will take a task to accomplish. Agent i then takes the current task area and determines what are the smaller and larger dimensions of this task area. Then, agent i moves to the middle of the larger dimension and determines how many sub areas will be generated by traversing across the line that splits the task area. The sub areas are determined by the holes encountered through this line, such that holes touching the line and the boundaries of the original area determine new sub areas. We give an example of this step in Figure 3, as explained below.

The reason for the room split is to parallelize the exploration task of the room, as the different agents will move to the various identified sub areas and explore them in parallel. The created sub areas are pushed to the stack, and each of them has the potential to be subdivided further.

Algorithm 3: Leader part of RoomScan

```
/* Algorithm performed by each Leader until room is entirely explored                    */
Upon (agent i returning to meeting point with Leader)
set agent i to Inactive;
if agent i has a stack or Leader has a stack then
    if agent i has a stack then
        Leader takes stack of newly discovered areas from agent i and merges with current stack;
    end
    if agent is work exceeds ratio 3 above Leader's work then
        Leader and agent i role swapped;
    end
    else if more than one agent is waiting without a stack then
        if agent is work is within ratio 4 of all other agents' work which is at least 1/4 the initial traversal of i then
            Leader splits stack into two parts of equal size;
            Leader gives one part back to agent i;
            Other part is given to an Inactive agent j with lowest work;
            Reset agent i work threshold to minimum;
            Signal agents i and j to get Active;
        end
        else
            Give stack to Inactive agent with lowest work;
        end
    end
    else if agent i within ratio 4 of all other Active agents then
        Give stack to agent i;
        Signal agent i to get Active;
    end
end
```

For example, Figure 3 shows the vertical traversal line that goes through the room and splits it across the hole boundaries (holes labeled 1, 2, 3) into three subareas A, B, and C. As shown in Figure 3, agent i does not have to change its traversal due to hole 1 as this hole is at least one point away from the traversal line. Agent i will have to change its traversal, due to hole 2 as this is along the traversal path and unknown to agent i. As Figure 3 has the Leader in the top right and the split is vertical, the agent pushes areas onto the stack from the right to the left so that the area furthest from the Leader is the next task to be explored. For each room on the right of the split the agent i pushes the furthest from the Leader on the stack, from Figure 3 only C is on the right. Then, agent i begins pushing the rooms on the left from the closest to the furthest from the Leader, Figure 3 first A then B. This process is similar for other locations of the Leader and split type, to maintain a stack of connected areas with the top being the furthest from the Leader on the opposite half of the area.

When the work threshold was reached and agent i still has a stack of task(s), it doubles the work threshold and goes back to the Leader. The purpose of doubling the work threshold is such that if no other agent was there to share work, then the other agents j are all Active and the agent i should perform more exploration before going to the Leader again. As all other agents j are Active, agent i can perform double the work and maintain the ratio 4 between all other Active agents j. If at any point agent i has no more tasks to do then agent i returns to the Leader.

Algorithm 4: Active agent i in RoomScan

```
/* Algorithm performed by each agent i after being activated by Leader in RoomScan */
// Initially work threshold is set to minimum by Leader
Upon(Signal received from Leader)
while Area.top && NewWork < workthreshold do
    CurArea = Area.pop;
    SmDim = min(CurrArea_x, CurrArea_y);
    LgDim = max(CurrArea_x, CurrArea_y);
    Move to middle of LgDim;
    Determine the path to be taken and the amount of areas, area_1, ..., area_i to be created given known holes along the
      path;
    Move along path to other appending NewWork end of area;
    If a hole is encountered agent i moves around to remain in their half of LgDim returning to path;
    Without loss of generality, assume Leader is top right of room and initial split is vertical;
    Push areas onto stack from right to left;
    for each right room do
        Push farthest from Leader first
    end
    for each left room do
        Push closest to Leader first
    end
    Similar stack construction for horizontal split and other locations of Leader;
end
Double work threshold and go to the Leader position;
```

5. Analysis

We continue now with the analysis of the algorithms. We have separated the two types of agents' actions, perimeter and room scan, to clearly discuss the work performed during each algorithm. Section 5.1 goes though the proof for perimeter scan and obstacle check. Section 5.2 will deal with room scan and the actions of the Leader and other agents (Active/Inactive). Finally, in Section 5.3, we combine the analysis for the works completed during all algorithms to prove the total work performed by each agent is $O(N/k)$ and total time is $\Theta(N/k)$ where N is the total number of grid points in the orthogonal region.

5.1. Perimeter Scan

To analyze these algorithms, we begin by proving the completeness of perimeter scan. We show that the algorithm performs a post-order traversal of all the rooms through converting the sequence of steps as rooms are generated into a tree structure. From here, we analyze the total amount of work that has been completed by each agent during the perimeter scan and obstacle check algorithms.

5.1.1. Sequence of Steps

Perimeter scan can be broken down into a sequence of steps $S = \sigma_1, \sigma_2, \ldots, \sigma_z$, such that each σ_a is a grid point $\sigma_a = (x_a, y_a)$, and any two consecutive points in the sequence differ in only one dimension, namely either $|x_a - x_{a+1}| = 1$ and $y_a = y_{a+1}$, or $|y_a - y_{a+1}| = 1$ and $x_a = x_{a+1}$, for $1 \leq a < z$. From this sequence, three types of subsequences occur: Wall, Corner, and Cross.

- Wall: subsequence of steps $s_a, \ldots, s_{a+b}$, where $j \geq 2$, such that either the x-coordinate is the same ($x_a = x_{a+1} = \cdots = x_{a+b}$), or the y-coordinate is the same ($y_a = y_{a+1} = \cdots = y_{a+b}$), i.e., the subsequence of steps follows along the same x or y coordinate.
- Corner: Is a single point σ_a, $1 \leq a \leq z$ that changes from the dimension of the subsequence path, either $|x_{a-1} - x_a| = 1$, $y_{a-1} = y_a$ and $|y_a - y_{a+1}| = 1$, $x_a = x_{a+1}$ or $|y_{a-1} - y_a| = 1$, $x_{a-1} = x_a$ and $|x_a - x_{a+1}| = 1$, $y_a = y_{a+1}$.
- Cross: a subsequence of closing a room from corner or a wall to another wall or corner. Namely, it is a subsequence after a corner or wall, $\sigma_a, \ldots, \sigma_{a+b}$ such that σ_{a+b+1} is a recently visited grid

point and σ_a and σ_{a+b+1} share either x or y value ($x_a = x_{a+b+1}$ and $y_a \neq y_{a+b+1}$ or $y_a = y_{a+1}$ and $x_a \neq x_{a+b+1}$). The other points will differ by one point in the x or y direction such that $x_i - x_{i-1} = 0$ and $|y_i - y_{i-1}| = 1$ or $x_i - x_{i-1} = 1$ and $|y_i - y_{i-1}| = 0$ as the agents could have encountered holes along the Cross. Algorithm 1, when closing a room ends in *Success* (Lines 11–25).

5.1.2. Special Subsequences and Rooms

Define a new sequence $Q_0 = q_1, q_2, \ldots, q_l$, where each element $q_i \in Q_0$ is a subsequence of S, representing either walls, corners, crosses, i.e., $q_i.type \in \{Wall, Corner, Cross, Empty\}$. First three type have an attribute: for *Wall*, $q_i.attribute \in \{s_x, s_y, e_x, e_y\}$, where the start point is (s_x, s_y), and the end point is (e_x, e_y); for *Corner*, $q_i.attribute \in \{x, y, Turn\}$, where (x, y) is the corner coordinate, and *Turn* denotes the direction of the turn which is either left or right; and for *Cross*, $q_i.attribute \in \{s_x s_y, e_x, e_y\}$, where start point is (s_x, y_x) and the end point is (e_x, e_y).

In closing a room, a room starts with a *Wall* or *Corner*, and ends with a *Cross*. A *Wall* is followed by a *Corner* or a *Cross* if they all had exterior walls along the path, or a *Cross* if there was a successful room closure direction. In Figure 4, part a begins with a *Wall*, q_i, and repeats until the *Corner* is reached that reached that has the previously explored point in q_i that was successfully reached by the front agent, whereas in part b, the room begins with a *Corner*, q_i.

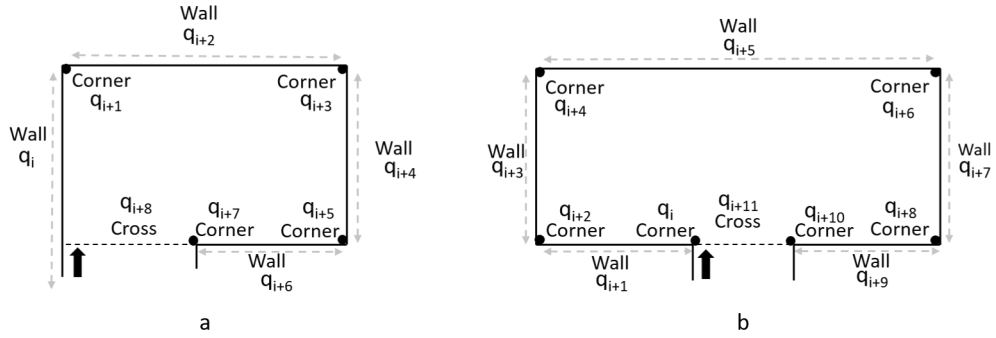

Figure 4. Two Basic Room Types. (**a**) Room that begins with a Wall; (**b**) Room that begins with a Corner.

5.1.3. Replacing Rooms and Sequence Transformation

We can define a sequence of subsequences $Q_0, Q_1, \ldots, Q_r$. Each pth subsequence is generated by the $p-1$ item in the sequence, so that subsequence Q_p is generated after room discovered in Q_{p-1}. Subsequence Q_p replaces the subsequence of Q_{p-1}: $q_i, q_{i+1}, \ldots, q_{i+j}$ that made up room R_p with a new subsequence that transforms the *Cross* into *Wall*(s) and *Corner*(s) depend on how the room was entered and the holes along the *Cross*. When a room is closed we want to treat the *Cross* of that room as a *Wall* for future room generation. To do this we create a new subsequence and treat that q_{i+j} as a *Wall* in the subsequence.

This new subsequence is $q'_i, q'_{i+1}, \ldots, q'_{i+j}$ such that j is the maximum number of subsequence elements. In Figure 5a the sequence in Q_i, $q_i, q_{i+1}, \ldots, q_{i+j}$ is replaced in Q_{i+1} to $q'_i, q'_{i+1}, q'_{i+2}, q'_{i+3}$, the attributes developed from the attributes of q_i, q_{i+j}, and q_{i+j-1}, see for example Table 1. Similarly, Figure 5b transforms from q_i, q_{i+j}, and q_{i+j-1} to q'_i, q'_{i+1}, q'_{i+2}. In performing this transformation, Table 1 depicts how each basic q is mapped to q'. Similarly, more complex q' are generated in the same manner. As these are only two basic room types, without any holes, other closures will have multiple q' created for q_{i+j}, the *Cross*, as with the introduction of holes the *Cross* will transform into multiple $Wall(s)$ and $Corner(s)$ as it is no longer a traversal along a single dimension.

For Q_r, the last room in the region, as $q_i, q_{i+1}, \ldots, q_{i+j}$ will be all elements of Q_{r-1}, Q_r replaces the subsequence of Q_{r-1}, $q_i, q_{i+1}, \ldots, q_{i+j}$ that made up room R_{r-1} with $q'_i.type = Empty$ signifying that the region is explored in its entirety.

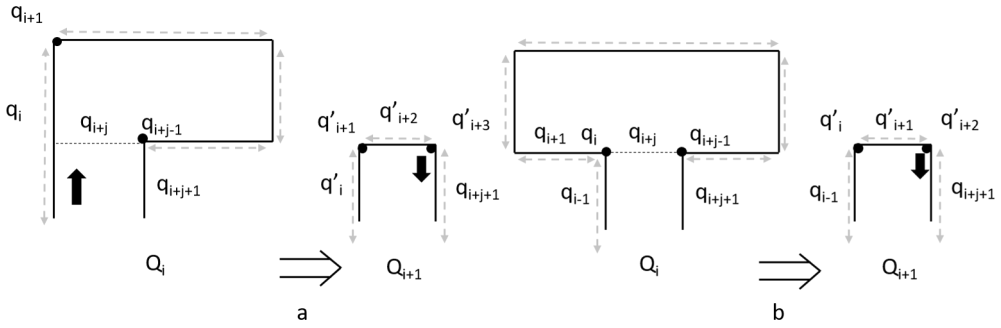

Figure 5. Two Basic Transformations After Room Scan. (**a**) Sequence after closure when starting with a Wall; (**b**) Sequence after closure when starting with a Corner.

Table 1. Basic Possible Values of q'.

	Figure 5a	Figure 5b
q'_i	$q'_i.e_y = q_{i+j}.e_y - 1$ $q'_i.s_y = q_i.s_y$ $q'_i.s_x = q_i.s_x$	$q'_i.e_y = q_{i+j}.e_y - 1$ $q'_i.s_y = q_{i-1}.s_y$ $q'_i.s_x = q_{i-1}.s_x$ $q'_i.e_x = q_{i-1}.e_x$
q'_{i+1}	$q'_i.x = q_{i+j}.e_x$ $q'_i.y = q_{i+j}.e_y - 1$	$q'_i.x = q_{i+j}.e_x$ $q'_i.y = q_{i+j}.e_y - 1$
q'_{i+2}	$q'_i.s_y = q_{i+j}.e_y - 1$ $q'_i.e_y = q_{i+j}.s_y - 1$ $q'_i.s_x = q_{i+j}.e_x$ $q'_i.s_x = q_{i+j}.e_x$	$q'_i.s_y = q_{i+j}.e_y - 1$ $q'_i.e_y = q_{i+j+1}.s_y$ $q'_i.s_x = q_{i+j}.e_x$ $q'_i.e_x = q_{i+j+1}.e_x$
q'_{i+3}	$q'_i.x = q_{i+j}.s_x$ $q'_i.y = q_{i+j}.s_y - 1$	

5.1.4. Conversion to Tree

Given Q_i is a sequence of steps q_i that discovers a room $i+1$, traversing the discovery of rooms in reverse, $Q_r, Q_{r-1}, \ldots, Q_1, Q_0$, a tree T of rooms can be generated (see Figure 6b). The root of T is room R_r (room R_3 in Figure 6) which is the last room to be discovered and corresponds to the room where the agents started the whole discovery process. Suppose that R_r corresponds to sequence $q_1, \ldots, q_l$ in Q_{r-1}. Room R_{r-1} was generated from Q_{r-2} and is included in $q_1, \ldots, q_l$.

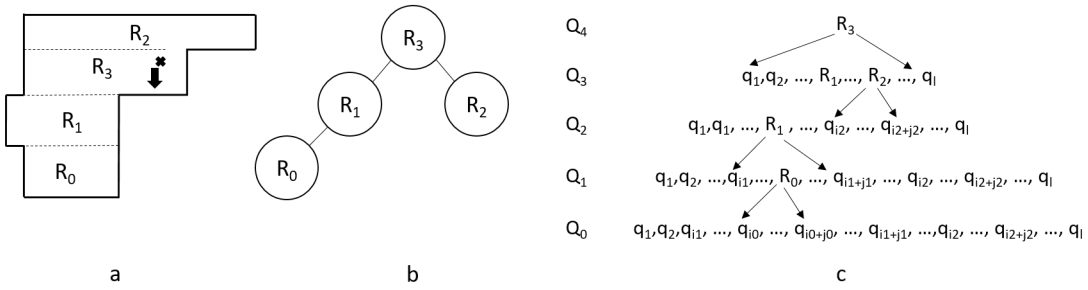

Figure 6. Conversion from Rooms to Tree Structure. (**a**) Sequence of room traversals; (**b**) Converting area to visualize as tree structure; (**c**) Changing sequence of points as rooms are discovered through traversal.

The children of R_i are the rooms R_h such that R_h is in the sequence of R_i in Q_{i-1} between the points q_i and q_{i+j} that generated R_i (see Figure 6c). Given multiple children of R_i they are added to the tree from left to right following their ordering in the sequence. If a room R_h does not have any room

sequence between its respective q_i and q_{i+j}, then it is considered to be a leaf of the tree. Following these rules we generate a tree structure, with R_r as the root, as we work backwards from the room discovery, Figure 6c, adding the respective children of each room as it is expanded from Q_i to Q_{i-1} until we return to Q_0.

For the discussion below assume that two points q_i and q_j are ordered as $q_i < q_j$ if $i < j$, and $q_i = q_j$ if $i = j$. Consider two rooms R_p and R_t that in Q_0 have the corresponding subsequences $q_{i_{0p}}, \ldots, q_{i_{0p}+j_{0p}}$ and $q_{i_{0t}}, \ldots, q_{i_{0t}+j_{0t}}$ respectively. The sequences of R_p and R_t in Q_0 are either *disjoined*, when $q_{i_{0p}} > q_{i_{0t}+j_{0t}}$ or $q_{i_{0p}+j_{0p}} < q_{i_{0t}}$, or *contained*, when $q_{i_{0p}} > q_{i_{0t}}$ and $q_{i_{0p}+j_{0p}} < q_{i_{0t}+j_{0t}}$ or $q_{i_{0t}} > q_{i_{0p}}$ and $q_{i_{0t}+j_{0t}} < q_{i_{0p}+j_{0p}}$, or *connected*, when $q_{i_{0p}} = q_{i_{0t}+j_{0t}}$ or $q_{i_{0t}} = q_{i_{0p}+j_{0p}}$.

Lemma 1. *Any pair of connected rooms, with neither being the root, have the same parent.*

Proof. Let us assume that R_p and R_u are two connected rooms. Let R_t be the parent R_p. Since R_t is the parent of R_p we have that R_t is the first room that contains R_p, we have $q_{i_{0t}} < q_{i_{0p}}$ and $q_{i_{0p}+j_{0p}} < q_{i_{0t}+j_{0t}}$. Also, from connected we know that either $q_{i_{0p}} = q_{i_{0u}+j_{0u}}$ or $q_{i_{0u}} = q_{i_{0p}+j_{0p}}$. Thus, $q_{i_{0t}} < q_{i_{0u}+j_{0u}}$ or $q_{i_{0u}} < q_{i_{0t}+j_{0t}}$. Therefore, from our projections R_t and R_u are not disjoined or connected, namely one contains the other.

Suppose that $t < u$. Therefore, R_u contains R_t. Since R_t contains R_p, we have that R_u contains R_p, which is a contradiction, since R_u and R_p are connected. Therefore, $t > u$. Consequently, R_t must also contain R_u.

Suppose that there is another room R_γ that is the parent of R_u. Therefore $q_{i_{0\gamma}} < q_{i_{0u}}$ as rooms R_u and R_p are connected then R_γ contains R_p, $q_{i_{0\gamma}} < q_{i_{0p}}$ As R_t is the parent of R_p, R_t must be contained in R_γ. This results in a contradiction because R_t contains R_u and is a descendant of R_γ. From this, R_t is the parent of R_p and R_u. □

If the room discovered in Q_{r-1} was not the room the agents began the exploration in, then there are two connected rooms that are the roots of two trees.

Lemma 2. *For disjoint rooms R_p and R_t where $p < t$, the subsequence that generates R_p must appear earlier than R_t in Q_0.*

Proof. Let's assume that it is possible for R_t subsequence, $q_{i_{0t}}, \ldots, q_{i_{0t}+j_{0t}}$, to appear earlier than R_p subsequence, $q_{i_{0p}}, \ldots, q_{i_{0p}+j_{0p}}$, in Q_0; $q_0, \ldots, q_{i_{0t}}, \ldots, q_{i_{0t}+j_{0t}}, \ldots, q_{i_{0p}}, \ldots, q_{i_{0p}+j_{0p}}, \ldots, q_l$. As traversal is from left to right, at some Q_c such that $c < p$ the subsequence, $q_{i_{0t}}, \ldots, q_{i_{0t}+j_{0t}}$ would be traversed and then at Q_p the subsequence $q_{i_{0p}}, \ldots, q_{i_{0p}+j_{0p}}$ would be traversed. Thus, it is not possible for an earlier subsequence that makes up a disjoined room to be explored after a later disjoined room. □

All points will be covered by agents after the entire perimeter is traversed. Given Lemmas 1 and Lemma 2 the whole area is divided into room which are explored in a post-order traversal of T. As we see from Figure 6a, from the agent's start position and traversal direction, the agents discover room $0, 1, 2$, and 3 which after generating the tree structure, Figure 6b, traverses the left subtree, then the right subtree, and finally the root, a post-order traversal.

Corollary 1. *The area is divided into rooms through the traversal of the perimeter. Following in reverse order create a tree T of rooms which are explored following post-order traversal of the tree.*

5.1.5. Perimeter Traversal

During perimeter scan, all agents will perform the traversal along the perimeter of the outer wall. When agents reach a corner that is along a previously explored point, the front agent will attempt to close the room and return. If an obstacle is detected then that agent will try to determine if it is a hole, given the aspect ratio, if it is then that agent continues to attempt to close the room. The agent returns to other agents if it is a room or runs into a wall. From there, if it is a room, room scan will begin,

otherwise the agents will continue perimeter scan. Before continuing perimeter scan the agent that has the least amount of perimeter work completed, such that it has not attempted to close a room or has had the smallest attempt of all agents to close a room, switches places with the front agent.

Lemma 3. *A shortest path of adjacent points in the orthogonal region between any two points on a single dimension with Euclidean distance C from each other, has a maximum length of* $C + 2aC$.

Proof. Without loss of generality, let us assume that the start and end points are on the same horizontal line. Then we can form a path which traverses from the start to end consisting of several path segments:

$$\sum_a p_a + \sum_b w_b + \sum_c g_c.$$

Each p_a are the lengths of horizontal path segments on same line as start and end, w_b are the lengths of horizontal path segments along encountered holes parallel to the line between start and end, g_c are the lengths of vertical path segments of encountered holes that are perpendicular to the line between start and end. There are two g_c for every w_b as there is a g_c connecting p_a to w_b and g_{c+1} connecting w_b to p_{a+1}. Due to holes being rectangular, the sum of $\sum p_a + \sum w_b = C$.

We will continue to show $\sum g_c \leq 2a \sum w_b$. As a is the aspect ratio of the hole dimensions, then for each hole length w_b we can assume the longer length is the associated g_c and g_{c+1}, meaning that at most these can be aw_i larger. Thus, since each w_b has two associated g_c, $\sum g_c \leq \sum 2aw_b$. As stated above, $\sum p_a + \sum w_b = C$ then, since $\sum w_b \leq C$, $\sum g_c \leq 2aC$, thus, $\sum p_a + \sum w_b + \sum g_c \leq C + 2aC$. □

Suppose that the perimeter, not including *Crosses* attempted, up to discovery of room R_i has P_i traversable points. Each of these P_i points is traversed by all k agents. The total traversable points is N_{it}, then orthogonal region up to room i has a total interior traversable points N_i such that $N_i = N_{it} - P_i$ where we remove the perimeter points from the total points in the area.

Lemma 4. *Total perimeter work from all agents performed is* $O(kP_i + N_i)$.

Proof. As the perimeter is traversed by all agents, k agents perform P_i movements to traverse all points along the perimeter. During a *Cross* only the agent with the least perimeter total work will gather work. Holes and walls discovered by an agent during a *Cross* are known by all other agents when the crossing agent returns. Agents do not attempt to close a room if a previously discovered hole or wall is located along that path. For this reason, there are two opportunities to traverse a position in the room, horizontal and vertical crosses, and as such, 4 traversals over any position in both directions of the Cross. Thus, the total amount of work performed during is kP_i for perimeter movements and $4N_i$ for the attempted crosses. □

From Lemma 3, there is a *Cross* with maximum Euclidean distance $Cross_{\max}$ corresponds to a maximum path distance $C_{\max} = Cross_{\max} + 2aCross_{\max}$, when moving around holes. $C_{\max}$ is the largest distance that any agent traverses during the traversal of its assigned Cross.

Lemma 5. *Difference in perimeter work between any two agents is* $O(C_{\max})$.

Proof. All agents perform the perimeter traversals, so the difference in work appears when crosses are performed, as these are done by a single agent. Agents swap after completing a Cross if another agent has less work to allow them to perform the next Cross. Let us assume that an agent could exceed $C_{\max}$ work compared to other agents. For two agents i and j with respective works W_i and W_j, if $W_i > W_j + C_{\max}$ then $W_i > W_j$ before the Cross was taken, as agent i could gain at most $C_{\max}$ new work. This contradicts the algorithm as the agent with the lower work would perform a Cross. Hence, all agents are bounded by a difference in work of $C_{\max}$. □

Lemma 6 (Perimeter Work). *If $N_i \geq 4C_{\max}k$, the perimeter work of each individual agent is $O(P_i + N_i/k)$.*

Proof. From Lemma 4, we have that the total work is $O(kP_i + N_i)$ and Lemma 5 proves the max difference between any two agents is $O(C_{\max})$. As agents are within $C_{\max}$ work of one another, each agent must have work at most $(kP_i + 4N_i)/k + C_{\max} = P_i + 4N_i/k + C_{\max}$. With the assumption that $N_i \geq 4C_{\max}k$, this can be simplified to $P_i + 4N_i/k + (1/(4k))N_i$ which is $O(P_i + N_i/k)$. □

Lemma 7 (Perimeter Time). *With total number of Corners Γ the total perimeter time is at most $P + 6\Gamma C_{\max}$.*

Proof. Each agent follows along the perimeter moving one position each time step, given P perimeter points, takes P time to visit each perimeter position. At every *Corner* we have at most 3 directions that a room can be generated with a maximum of $C_{\max}$ distance to traverse. An agent must traverse the closure in both directions, thus $6\Gamma C_{\max}$ to traverse possible room closures for all *Corners* Γ. This creates an upper bound on the total time to finish the perimeter scan portion $P + 6\Gamma C_{\max}$. □

5.2. Room Scan

We continue with the work analysis of the agents during room scan. In our algorithm, as the perimeter scan progresses, new rooms are being discovered, and each room is explored separately in room scan by the work performed by the Active agents and controlled by the Leader. For balancing the work, the robots keep a separate count of the total work performed during room scan. The first room discovered is a special case, as there are no previous rooms or room work. Hence, we start with analyzing the work completed by each agent in the first room. From this point, we can now describe the relationship between all agents work throughout each subsequent room exploration to explain total work of all agents together and work performed by each agent individually after the completion of each separate room. Finally, after the last room we can analyze the total work and work completed by each agent at the end of the whole area exploration.

5.2.1. First Room Exploration

Consider the first closed room and suppose that it is of size $L_{long} \times L_{short}$, where the length of L_{long} is at least as large as L_{short}. Suppose also that there is a total of n traversable points in the room. The aspect ratio of the holes in the room is a. Without loss of generality, we will assume that L_{long} is horizontal, and L_{short} is vertical. All agents currently have 0 initial work. Before, we proceed with the work analysis, we need bound the longest shortest path in the room, which will be later used to bound the difference on the works of different agents.

Lemma 8. *The maximum shortest path between any two points in a room has length $D = O(aL_{long})$.*

Proof. A shortest path between any arbitrary pair of points in the room can be formed by connecting two subpaths h and v, the first is horizontal of length H, and the second is vertical of length V, to reach the other point.

There can be a hole along the two subpaths. This creates key positions in the path pos_0, pos_1, pos_2, and pos_3. pos_0 is the starting point, and pos_3 is the ending point. Assuming horizontal subpath taken first, pos_0 and pos_1 are the start and end positions, respectively, of h and pos_2 and pos_3 are the start and end positions of v. If $pos_1 \neq pos_2$ then there is a path around a hole that goes from $pos_1.x$ to $pos_2.x$ and $pos_1.y$ to $pos_2.y$. This hole is only traversed along two dimensions, one horizontal and one vertical, with lengths l_1 and l_2, respectively. Therefore, $l_1 + l_2 \leq 2L_{long}$.

From Lemma 3, $H \leq (2a+1)L_{long}$ and $V \leq (2a+1)L_{short}$. Thus, the total length of the path at most $(2a+1)L_{long} + (2a+1)L_{short} + l_1 + l_2 \leq (4a+4)L_{long}$. □

Lemma 9. *In a room with n points the total work by all agents is $O(n)$.*

Proof. Let us take the case that there is a single agent with no specific work threshold to returning to the meeting point in the room. From Algorithm 4, the agent takes the original area, moves to the middle of the long dimension and performs a split traversing unexplored positions along this traversal. After exploration $Area_0$, the total room, generates a stack of connected areas $Area_1, Area_2, \ldots, Area_i$ such that there is a path of previously explored points on the split that connects each pair of $Area_j$ and $Area_{j+1}$. When $Area_j$ is popped off it can either make sub areas $Area_{j_1}, \ldots, Area_{j_i}$ that are connected through the split of $Area_j$ and $Area_{j_i}$ connect to $Area_{j+1}$ through the previous split or the agent moves to $Area_{j+1}$ along the split of $Area_0$. Through this recursive process, the agent traverses split positions at most 4 times (down and up both halves). Every point with $Area_0$ will be part a split in one of the recursive steps. Therefore, the whole recursive process is generating $4n$ total traversals of the total $Area_0$.

Now let us add multiple agents that balance their work, but without acting concurrently, i.e., when the work threshold is reached an agent swaps its stack of work with another agent to balance its work. With a threshold, agents are now able to leave and return to an $Area_j$ if they have completed enough work to return to the Leader and another agent returns in their place. From Lemma 8, the threshold $2D$ newly explored points. In this, $2D$ newly explored points pays for $2D$ already explored points when going to the origin and swap the stacks. Since in the worst case an agent may return at most $n/2D$ times to the origin, the total work with the threshold is at most $4n + (n/2D)2D \leq 4n + n \leq 5n$.

Finally, we view the complete algorithm with multiple agents that can also share the work when the threshold is reached, i.e., the agents perform parallel exploration. With agents splitting their work with another agent, according to Algorithm 4 one agent had work $Area_1, Area_2, \ldots, Area_i$ and reached the threshold. Upon returning, it shares half the work, some area $Area_j$, such that the first agent contains $Area_1, Area_2, \ldots, Area_j$ and the second agent contains $Area_{j+1}, \ldots, Area_i$. With this addition, agents can perform work in parallel, gaining the ability to share work but requiring another D each time sharing happens to account for the new agent to traverse to its given area. Therefore, total work becomes at most $4n + (n/2D)3D = 4n + 3n/2 \leq 6n$. □

Lemma 10. *At the end of the room scan of a room with n points, given the initial work of all agents is* 0*, one of the two holds:*

i. If $n < 2Dk$, the difference in work between any two agents is $4D$*, or otherwise,*
ii. the ratio of the work between any two agents is bounded by 8*.*

Proof. For case i, as agents return to meet after $2D$ work, where $n < 2Dk$, then there are not enough traversable points for all agents to get work. This will lead to some out of k agents completing $4D$ work and other agents not completing any work.

For case ii, $n \geq 2Dk$, such that all agents were given a chance to be Active. The Leader maintains a ratio 4 of work between all agents and a ratio 3 between all other agents and the Leader. As two agents may approach the ratio 4 and the lower one could complete work just after the one with greater work has completed, then the agent with greater work will double their current work before returning being at most ratio 8 more work than this agent. After returning this agent will be set to Inactive and swapped with the lower total work agent. As the Leader has global knowledge of the work of all agents, after they share information, the Leader can maintain the ratio 8 of work between all agents. □

Theorem 1. *At the end of the room scan with n points, each agent performs $O(n/k)$ work when $n \geq 2Dk$.*

Proof. From Lemma 9, let the total work be $W = c_1 n$, for some constant $c_1 > 0$. From case (ii) of Lemma 10, the ratio of the work between any two agents is at most 8. Therefore, since there is an agent that has work at most $c_1 n/k$, each agent's work does not exceed $8c_1 n/k = O(n/k)$. □

5.2.2. Room Exploration

Let $R_1, R_2, \ldots, R_r$ be the sequence of rooms that are visited in the algorithm. For each room R_i, let D_i be the maximum shortest path distance between any pair of points in R_i, and let n_i be the total number of traversable points in the room. For each room, the total number of previously explored points in the previously explored rooms are taken into account when determining the difference in work between agents. As with perimeter scan, we will use $D_{\max} = \max_i D_i$. We will show that when the sum of work in all rooms explored exceeds $kD_{\max}$, a ratio 8 between the works of the agents is maintained. We define N_i as the sum of all the traversable points in all the rooms up to R_i, i.e., $N_i = \sum_{j=1}^{i} n_j$. After the completion of the room R_i, each agent k has an associated work w_k^i.

We extend Lemma 10, to the more general case where agents have prior work.

Lemma 11. *If before the scan of room R_i, $i > 1$, the work of any two agents g and h differ in work $|w_g^{i-1} - w_h^{i-1}| \leq 4D_{\max}$, then at the end of the room scan of R_i then the difference is maintained or a ratio 8 is achieved.*

Proof. Initially, only one agent gets Active in R_i, suppose g. It must be that the work of g before R_i, w_g^{i-1}, is at most the work of every other agent h. In its traversal, g gains at most $4D_i \leq 4D_{\max}$ additional work. Therefore, $|(w_g^{i-1} + 4D_i) - w_h^{i-1}| \leq |(w_g^{i-1} + 4D_{\max}) - w_h^{i-1}| \leq 4D_{\max}$. Thus, since our algorithm always picks an agent with the lowest work, as long as a single agent is scanning the room R_i, the maximum work difference of $4D_{\max}$ is preserved.

Now, that the algorithm sends two or more agents for scanning. The agents that are selected by the Leader to scan the room must start with a ratio of work 4 within each other. These selected agents have also the lowest work at the time of their activation, with respect to all agents k, and each agent must have minimum total work of D_i. At any time during their scan, the Active agents accumulate new work, but each agent at most doubles its work (according to our algorithm). Therefore, the ratio of work among those Active agents does not exceed 8.

Between Active and Inactive agents, the difference of work is decreasing until the time when Inactive agents get less work than the Active, and the Inactive become Active themselves. Until that happens the difference of work between Active and Inactive is $4D_{\max}$. After that, the ratio 8 is also maintained between Active and Inactive agents. Even if all agents get Active, then they have a ratio 4 among them when they start scanning and will maintain a ratio 8 upon returning if one becomes Inactive due to the completion of their area. Therefore, in all cases, if multiple agents start scanning either the work difference is bounded by $4D_{\max}$ or a ratio 8 between the works of the agents is maintained. □

Lemma 12. *If before the scan of room R_i, $i > 1$, the work of any two agents g and h have a difference in their ratio of work $1/8 \leq w_g^{i-1}/w_h^{i-1} \leq 8$, then at the end of the room scan of R_i then ratio 8 is maintained or regress to difference of $4D_{\max}$.*

Proof. Similar to the proof of Lemma 11, at the beginning only one agent scans R_i, say agent g, which gains at most $4D_i$ work. Any other agent h must have work $w_h^{i-1} \geq w_g^{i-1}$. If $w_h^{i-1} < D_i/4$, then, the ratio 8 between these two agents is not preserved, but there is a maximum difference of $4D_{\max}$ in their works.

If more than one agent becomes Active for scanning, then similar to the proof of Lemma 11, the work ratio of 8 is maintained or the difference of works is bounded by $4D_{\max}$. □

Lemma 13. *At the end of scanning room R_i for any two agents it holds that either $|w_g^i - w_h^i| \leq 4D_{\max}$ or $1/8 \leq w_g^i/w_h^i \leq 8$.*

Proof. We will use induction. For the base case, $i = 1$, the claim holds immediately as initially no agent has any work so all are within $4D_{\max}$ and ratio 8. Suppose the claim holds up to the scanning

of room R_{i-1}. By induction hypothesis, for any two agents g and h, $|w_g^{i-1} - w_h^{i-1}| \leq 4D_{\max}$ or $1/8 \leq w_g^{i-1}/w_h^{i-1} \leq 8$.

If $|w_g^{i-1} - w_h^{i-1}| \leq 4D_{\max}$, then from Lemma 11, at the end of the scanning of R_i the difference of the works between two agents is either bounded $4D_{\max}$ or the ratio of their works is at most 8. Similarly, if $1/8 \leq w_g^{i-1}/w_h^{i-1} \leq 8$, then from Lemma 12, the work difference is bounded by $4D_{\max}$ or the ratio of their works is at most 8.

Therefore, in all cases it holds $|w_g^i - w_h^i| \leq 4D_{\max}$ or $1/8 \leq w_g^i/w_h^i \leq 8$, as needed. □

Lemma 14 (Room Work)**.** *Upon the completion of each room N_i the total work done up to that point is $O(N_i)$.*

Proof. From above, $N_i = \sum_{j=1}^{i} n_j$, and from Lemma 9, the work done in a room is $O(n_j)$, more specifically at most $6n$. The sum of work for all rooms becomes at most $6n_1 + 6n_2 + \ldots + 6n_i = 6N_i$ which is $O(N_i)$. □

Lemma 15. *Each agent performs $O(N_i/k)$ work when $N_i \geq 2D_{\max}k$.*

Proof. From Lemma 14 the total work from all the agents combined is $O(N_i)$, say $c_1 N_i$ for some constant c_1. Therefore, some agent has performed work at most $c_1 N_i/k$. From Lemma 13, for any two agents g and h we have, $|w_g^i - w_h^i| \leq 4D_{\max}$ or $1/8 \leq w_g^i/w_h^i \leq 8$. Suppose that g has the maximum total work, and h has the minimum total work, among all k agents.

If $|w_g^i - w_h^i| \leq 4D_{\max}$, then agent has g has no more work than $c_1 N_i/k + 4D_{\max} \leq c_1 N_i/k + 2N_i/k = O(N_i/k)$. If $1/8 \leq w_g^i/w_h^i \leq 8$, then agent g has no more work than $8c_1 N_i/k = O(N_i/k)$. Hence, in any case each agent's work is $O(N_i/k)$. □

To perform analysis of the time complexity of room scan, we examine the total time given the orthogonal area contains no holes. (Logarithms are base 2.)

Lemma 16 (Room Time)**.** *Assuming no holes, upon the completion of each room R_i the maximum total time up to that point is $\sum_{i=1}^{r}((6n_i - 4D_i \log k)/(k-1) + 4D_i \log k) + 4D_{\max}k$.*

Proof. At the start of room scan, we have an initial swapping of work until every agent reach $4D_{\max}$ work, creating a total time of at most $4D_{\max}k$ for each agent to swap to be Active. After this point, at the beginning of each room, as we perform slow start, the agents share work doubling the number of Active agents each $4D_i$ time steps taking a total of $4D_i \log k$ time for all agents to become Active. As there are no holes, the algorithm generates rectangular areas initially and all agents get an equal portion of the area. Now that all agents are Active, and given that the work performed in a room is $6n_i$ (Lemma 9) we subtract the amount of work that has been performed during the sharing process and divide by the number of Active agents, $k-1$ as one agent is Leader, achieving the maximum total time for room completion $\sum_{i=1}^{r}((6n_i - 4D_i \log k)/(k-1) + 4D_i \log k) + 4D_{\max}k$. □

5.3. Combination of Work

In this section, we will combine Lemmas 4 and 14 to get the total work for all movements in the algorithm. We will also combine the total amount of work completed by an agent to get the total work done by each agent using Lemmas 6 and 15.

For the combination of these Lemmas and to use Lemma 15 for work of agents in room scan we assume $N_i \geq 2D_{\max}k$. As $C_{\max}$ is along a single dimension and $D_{\max}$ is along two dimensions, $D_{\max} \geq 2C_{\max}$. From Lemma 4, we get that the total work during the perimeter is $O(kP_i + N_i)$, and from Lemma 14 we get that the total work during all room scans combined is $O(N_i)$. Therefore,

Corollary 2. *The total work done by all agents at the end of scanning room R_i is $O(kP_i + N_i)$.*

From Lemma 6 agents perform work $O(P_i + N_i/k)$ and from Lemma 15 each agent performs work $O(N_i/k)$. Therefore,

Corollary 3. *At the end of scanning room R_i, each agent performs work $O(P_i + N_i/k)$.*

Theorem 2 (Work Balancing)**.** *If $P_i = (N_i/k)$, then at the end of scanning room R_i, the total work of all agents is $O(N_i)$ and the work of each agent is $O(N_i/k)$.*

Proof. Since from Corollary 2 the total work is $O(kP_i + N_i)$, if $P_i \leq c_1(N_i/k)$ for some constant c_1, then $O(kP_i + N_i) = O(N_i)$. From Corollary 3, the total work performed by each individual agent is $O(P_i + N_i/k)$, with the above value of P_i this is $O(N_i/k)$. □

5.4. Combination of Time

Given the Lemmas 7 and 16 we will show the total time to complete the exploration.

Corollary 4. *The total time to complete exploration at the end of room scan is $P + 6\Gamma C_{\max} + 4D_{\max}k + \sum_{i=1}^{r}((6n_i - 4D_i \log k)/(k-1) + 4D_i \log k)$.*

Given this total time to complete the exploration we have certain areas in which the total time is $\Theta(N/k)$. Any exploration cannot do better than $\Omega(N/k)$ as this is all agent would be actively exploring during the entire exploration. For our exploration, we define a large room i is such that $4D_i \log k$ is of order $(6n_i - 4D_i \log k)/(k-1)$.

Corollary 5 (Parallel Exploration)**.** *When in an orthogonal area is made up of a majority of large rooms, and the terms P, $6\Gamma C_{\max}$, and $4D_{\max}k$ are $O(N/k)$, then time complexity is of $\Theta(N/k)$, which is asymptotically optimal.*

6. Conclusions

This paper proposes multiple algorithms, to perform exploration and a slow start, for multiple agent to perform parallel exploration with a limited communication range. Through the introduction of unknown holes in the area the room exploration algorithm of Leader with slow start was developed to minimize the cost of returning to the Leader while maintaining near equal work. Through exponential back-off with a reset to communicate with the Leader, communication costs are decreased as well as the cost of returning to communicate. With our algorithms, we achieve an overall performance of $O(N/k)$ agent work in asymptotically optimal time $\Theta(N/k)$.

Our future work is to implement the algorithms on several agents. We also plan to explore the algorithm with agent failure. Lastly, it is our goal to remove the orthogonal region and rectangular hole constraints to develop algorithms that enable the agents to explore concave polygons.

Author Contributions: Conceptualization, W.C., C.B., L.P., D.S. and H.-C.W.; Formal analysis, W.C. and C.B.; Investigation, W.C., C.B., L.P., D.S. and H.-C.W.; Methodology, D.S.; Supervision, C.B. and H.-C.W.; Validation, L.P.; Writing—original draft, W.C.; Writing—review & editing, W.C.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Alam, T.; Bobadilla, L.; Shell, D. Minimalist Robot Navigation and Coverage using a Dynamical System Approach. In Proceedings of the 2017 First IEEE International Conference on Robotic Computing, Taichung, Taiwan, 10–12 April 2017.
2. Galceran, E.; Carreras, M. A survey on coverage path planning for robotics. *Robot. Auton. Syst.* **2013**, *61*, 1258–1276. [CrossRef]

3. Pierson, A.; Figueiredo, L.C.; Pimenta, L.C.A.; Schwager, M. Adapting to performance variations in multi-robot coverage. In Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, 26–30 May 2015; pp. 415–420. doi:10.1109/ICRA.2015.7139032. [CrossRef]
4. Gasparri, A.; Krishnamachari, B.; S. Sukhatme, G. A framework for multi-robot node coverage in sensor networks. *Ann. Math. Artif. Intell.* **2008**, *52*, 281–305. [CrossRef]
5. Batalin, M.A.; Sukhatme, G.S.; Hattig, M. Mobile robot navigation using a sensor network. In Proceedings of the IEEE International Conference on Robotics and Automation, New Orleans, LA, USA, 26 April–1 May 2004; Volume 1, pp. 636–641. doi:10.1109/ROBOT.2004.1307220. [CrossRef]
6. Freeman, J. Robot Assisted Wireless Sensor Network for Monitoring and Detection of Explosives in Indoor Environment. *Int. J. Comput. Sci. Eng.* **2011**, *3*, 2046–2053.
7. Kumar, V.; Rus, D.; Singh, S. Robot and Sensor Networks for First Responders. *IEEE Pervas. Comput.* **2004**, *3*, 24–33. [CrossRef]
8. Masoud, A.; Al-Shaikhi, A. Time-Sensitive, Sensor-based, Joint Planning and Control of Mobile Robots in Cluttered Spaces: A Harmonic Potential Approach. In Proceedings of the 54th IEEE Conference on Decision and Control (CDC), Osaka, Japan, 15–18 December 2015 .
9. Reich, J.; Sklar, E. Robot-Sensor Networks For Search And Rescue In Proceedings of the IEEE International Workshop on Safety, Security and Rescue Robotics, Gaithersburg, MD, USA, 22–24 August 2006.
10. Ortolf, C.; Schindelhauer, C. Online multi-robot exploration of grid graphs with rectangular obstacles. In Proceedings of the Twenty-Fourth Annual Acm Symposium On Parallelism In Algorithms And Architectures, San Diego, CA, USA, 29 June–1 July 2012; pp. 27–36.
11. Ray, D.; Mandal, A.; Majumder, S.; Mukhopadhyay, S. Human-like Gradual Multi-agent Q-learning Using the Concept of Behavior-based Robotics for Autonomous Exploration. In Proceedings of the IEEE International Conference on Robotics and Biomimetrics, Karon Beach, Thailand, 7–11 December 2011.
12. Sheng, W.; Yang, Q.; Tan, J.; Xi, N. Distributed multi-robot coordination in area exploration. *Robot. Auton. Syst.* **2006**, *54*, 945–955. [CrossRef]
13. Sharma, R.; Honc, D.; Dušek, F.; Kumar T., G. Frontier Based Multi Robot Area Exploration Using Prioritized Routing. In Proceedings of the 30th European Conference on Modelling and Simulation, Regensburg, Germany, 31 May–3 June 2016; pp. 25–30.
14. Agrawal, V. Multi Robot Area Exploration using Modified Frontier Algorithm. *Int. J. Comput. Eng. Appl.* **2015**, *IX*, 1–8.
15. Fazli, P. Fault-tolerant multi-robot area coverage with limited visibility. In Proceedings of the ICRA 2010 Workshop on Search and Pursuit/Evasion in the Physical World: Efficiency, Scalability, and Guarantees, Anchorage, Alaska, 3–8 May 2010.
16. Palmieri, N.; Yang, X.; Marano, S. Coordination Techniques of Mobile Robots with Energy Constraints. In Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS), Montreal, QC, Canada, 24–27 July 2016 .
17. Saeedi, S.; Paull, L.; Trentini, M.; Seto, M.; Li, H. Group Mapping: A Topological Approach to Map Merging for Multiple Robots. *IEEE Robot. Autom.* **2014**, *21*, 60–72. [CrossRef]
18. Zhou, Y.; Zincir-Heywood, A. Intelligent Agents for Routing on Mobile Ad-hoc Networks. In Proceedings of the Conference on Communication Networks and Services Research, Fredericton, NB, Canada, 21 May 2004.
19. Rodriguez, A.; Gomez, J.; Diaconescu, A. Foraging-Inspired Self-Organisation for Terrain Exploration with Failure-Prone Agents. In Proceedings of the IEEE 9th International Conference on Self-Adaptive and Self-Organizing Systems, Cambridge, MA, USA, 21–25 September 2015.
20. Veseras, A.; Wiedemann, T.; Manss, C.; Magel, L.; Mueller, J.; Shutin, D.; Merino, L. Decentralized Multi-agent Exploration with Online-learning of Gaussian Processes. In Proceedings of the 2016 IEEE International Conference on Robotics and Automation (ICRA), Stockholm, Sweden, 16–21 May 2016 .
21. Fazli, P.; Davoodi, A.; Mackworth, A. Multi-robot repeated area coverage. *Auton. Robots* **2013**, *34*, 251–276. [CrossRef]
22. Flocchini, P.; Kellett, M.; Mason, P.; Santoro, N. Map Construction and Exploration by Mobile Agents Scattered in a Dangerous Network. In Proceedings of the IEEE International Symposium on Parallel and Distributed Processing, Rome, Italy, 23–29 May 2009.
23. Basilico, N.; Amigoni, F. Exploration strategies based on multi-criteria decision making for searching environments in rescue operations. *Auton. Robots* **2011**, *31*, 401–417. [CrossRef]

24. Roy, N.; Dudek, G. Collaborative Robot Exploration and Rendezvous: Algorithms, Performance Bounds and Observations. *Auton. Robots* **2001**, *11*, 117–136. [CrossRef]
25. Hoog, J.; Cameron, S.; Visser, A. Role-Based Autonomous Mulit-robot Exploration. In Proceedings of the 2009 Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, Athens, Greece, 15–20 November 2009.