Complete Distributed Coverage of Rectilinear Environments

Zack J. Butler, *Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA*
Alfred A. Rizzi, *Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA*
Ralph L. Hollis, *Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA*

Complete coverage of an unknown environment is a valuable skill for a variety of robot tasks such as floor cleaning and mine detection. Additionally, for a team of robots, the ability to cooperatively perform such a task can significantly improve their efficiency. This paper presents a complete algorithm DC_R (distributed coverage of rectilinear environments) which gives robots this ability. DC_R is applicable to teams of square robots operating in finite rectilinear environments and executes independently on each robot in the team, directing the individual robots so as to cooperatively cover their shared environment relying only on intrinsic contact sensing to detect boundaries. DC_R exploits the structure of this environment along with reliable position sensing to become the first algorithm capable of generating cooperative coverage without the use of either a central controller or knowledge of the robots' initial positions. We present a completeness proof of DC_R, which shows that the team of robots will always completely cover their environment. DC_R has also been implemented successfully in simulation, and future extensions are presented which will enable instantiation on a real-world system.

1 Introduction

The *coverage* problem, that of planning a path for a sensor, effector, or robot to reach every point in an environment, is one that appears in a number of domains. The problem of *sensor-based* coverage, that of planning such a path from sensor data in the absence of *a priori* information about the environment, is limited to robotics, but also applies to a number of different tasks. What is common to all these problems, whether a spray painting task on a known surface or a mine detection task with little or no initial information, is a need for assurance of complete coverage. For known areas, a path can be correctly generated off-line [1], but in the sensor-based case, the usual solution is instead to use a strict geometric algorithm about which completeness can be proven for any environment of a given class. Tasks which may be accomplished by multiple robots introduce additional complexity, since each point in the environment need only be reached by one of several robots, and in order to cooperate, the robots must know (or discover) each other's location. However, using multiple robots gives the potential for increased efficiency in terms of total time required.

While a number of sensor-based coverage algorithms have been proposed, in most, the algorithm begins by assuming the environment to be simply shaped (e.g. simply connected, monotone, convex, etc.). To cover its environment, the robot may then execute a simple coverage path until it discovers evidence that contradicts the initial assumption, at which point one of several strategies is used to ensure coverage on all sides of the newly discovered obstacle. An algorithm presented by Lumelsky et al. in [2] and extended in [3] produces coverage of $\mathcal{C}^2$ environments for robots with finite non-zero sensing radius by recursively building a subroutine stack to ensure all areas of the environment are covered. This algorithm does not explicitly build a map, in contrast to sensor-based coverage work by Acar [4] based on a planned coverage strategy outlined in [5]. In [4], a cellular decomposition of the environment is constructed and used to form a graph which in turn is used to plan coverage — when a specific cell has been covered, the robot uses the structure of the graph to plan a path to an unexplored area, and when the graph has no unexplored edges, coverage is complete. The cellular decomposition approach of [5] also inspired the algorithm presented in [6], which in turn is the basis for the current work. In [6], we presented an algorithm for coverage of rectilinear environments by a single robot using only intrinsic contact sensing. This algorithm also explicitly leveraged the degeneracies of the environment (degenerate in the $\mathcal{C}^2$ sense)

by decomposing the free space into a set of rectangular cells.

In contrast to the more commonly studied coverage tasks mentioned above, the inspiration for the current work comes from a manufacturing environment. The *minifactory*, an automated assembly system under development in the Microdynamic Systems Laboratory[1], has been built within a framework that provides for rapid design, programming and deployment [7]. A minifactory includes several types of independent robots, but this work concentrates on the *couriers*, small tethered robots that operate on a set of tileable *platens* which form the factory floor. The couriers have micron-level position sensing but only intrinsic contact sensing to detect the boundaries of their environment. In addition, each is equipped with an upward-pointing optical sensor to locate LED beacons placed on overhead robots as calibration targets. One of the tasks for the couriers is to collectively explore the as-built factory from unknown initial positions to generate a complete factory map. This task has led to the investigation of coverage algorithms for teams of robots, with the restrictive environment providing a simplified domain to consider. The algorithm developed here therefore applies to teams of square robots with intrinsic contact sensing operating in a shared, connected rectilinear environment with finite boundary and area. In addition, the robots in the team will not know their relative initial positions or orientations, however, due to the structure of the environment, their orientation will be one of four distinct values (i.e. with axes aligned with the environment boundaries) and cannot change.

Like the work described here, previous work in distributed robotics has presented the use of a common algorithm executed by each robot in a team (without a central controller) to achieve a specific task [8, 9, 10]. For example, in the work of Donald et al. [8], several distributed algorithms were presented to perform a cooperative manipulation task. There, however, the goal was to recast a simple provable algorithm in such a way that explicit communication was unnecessary, but could rather be implicit in the task mechanics. In our work, the environment is static, and so this reduction is not available, and the underlying algorithm (single-robot coverage) is much more complex. Other work on decentralized control of cooperative mobile robots has generally focused on the creation of a certain group behavior (as simple as foraging [9] or as complex as playing soccer [10]) without proof of the correctness or completeness of the individual or group algorithms.

[1] More information at http://www.cs.cmu.edu/~msl

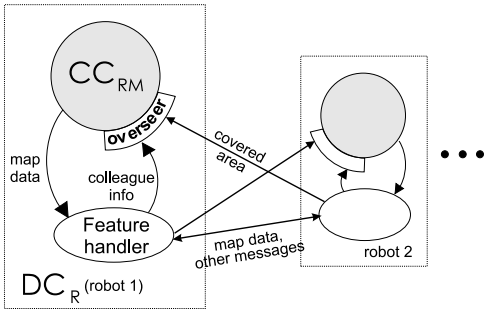

Figure 1: *A schematic rendition of the algorithm DC_R, a copy of which is run independently by each robot performing coverage.*

On the other hand, research into algorithms for complete coverage of an environment by cooperating robots has so far used a central controller deploying robots from known locations, which is not satisfactory for the minifactory problem. For example, Gage's work [11] uses random walks by a large team with a common home position to generate probabilistically complete coverage. A fairly abstract algorithm presented by Rao et al. [12] uses a small team of point-sized robots with infinite range sensing to build a visibility graph of a polygonal environment. In contrast, work by Rekleitis et al. [13] uses cooperating robots with mutual remote sensing abilities, but with explicit cooperation to reduce mapping errors rather than to increase efficiency.

2 DC_R: Overview

DC_R is an algorithm developed for square robots that use only intrinsic contact sensing, and produces complete coverage of any finite rectilinear environment while using cooperation between robots to produce coverage more efficiently. It consists of three distinct components, shown schematically in Fig. 1. The first, CC_{RM}, covers the environment by incrementally building a cellular decomposition $\mathbf{C}$ ($\mathbf{C} = \{C_0, \ldots, C_n\}$). It uses only $\mathbf{C}$ and the robot's current position $p = (p_x, p_y)$ (i.e. time-based history is not used) to direct the robot to continue coverage. CC_{RM} is based on the work in [6], and performs coverage without taking into account any other robots in its team. However, with the other components of DC_R properly designed,

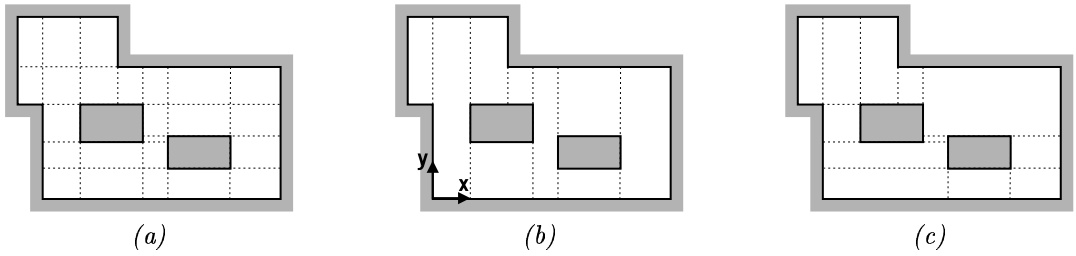

Figure 2: *Examples of (a) the sweep-invariant decomposition, (b) an oriented rectilinear decomposition, and (c) a possible generic rectilinear decomposition of a rectilinear environment.*

it performs coverage equally capably in a cooperative setting. The second part of DC_R, the *feature handler*, derives pre-specified types of features from **C** and communicates with other robots in the team to determine the relative position of the various robots. Finally, the *overseer* takes incoming data from colleagues and alters **C**. This is done without the explicit "knowledge" of CC_{RM}, but because CC_{RM} uses no state other than **C** and p, the overseer can alter **C** to incorporate the new data, and as long as this is done such that **C** remains in a state admissible to CC_{RM} (as described in Sec. 2.4), coverage continues.

2.1 Cell decompositions under DC_R

Before discussing the makeup of DC_R, it may be instructive to describe the type of cell decompositions that will be constructed. This discussion will focus only on the geometry of the cells in the decomposition, but in each type of decomposition, every cell also includes "connections" (geometric references) to each of its neighbors that allow for the straightforward creation of an adjacency graph of the decomposition. One way to uniquely decompose a known rectilinear environment is the *sweep-invariant* decomposition (SID), as shown in Fig. 2a. In a SID, each obstacle or boundary edge is extended until a perpendicular wall is reached, with all extended edges defining cell boundaries — as a result, a cell's edge may be determined by an arbitrarily distant boundary component. Thus, it is not possible to incrementally construct the SID without splitting cells that have already been completed, since determining a cell's extent could require knowledge of features that are arbitrarily distant from the cell itself. This difficulty of incremental construction makes sensor-based coverage based on the SID impractical. A different, nearly unique type of decomposition is the *oriented rectilinear* decomposition, shown in Fig. 2b. This decomposition is produced by extending all vertical edges (the x values of these edges are called *interesting points*, analogous to critical points of a sweep through a $\mathcal{C}^2$ environment) to form cell boundaries, and can be incrementally constructed and easily covered with the use of seed-sowing paths as described below. This decomposition is produced by DC_R when performing coverage without colleagues.

Multiple robots performing cooperative coverage in a shared environment will not necessarily have the same orientation, and therefore will not necessarily build the same oriented rectilinear decomposition. This means that neither of the above decompositions can be built in a cooperative setting. Instead, the cell decompositions that are constructed under DC_R fall into a class referred to here as *generic rectilinear* decompositions (GRDs), an example of which can be seen in Fig. 2c. A GRD consists of cells that are rectangular and supersets of cells of the SID. In addition, in a *valid* GRD, no two cells have overlapping area ($\forall i, j : C_i \cap C_j = \emptyset$) and all cells contain connections to all neighbors across their common edges. It should also be noted that during the performance of DC_R, a valid GRD may consist of multiple disconnected components. This is because robots only share completed cells, but may meet in an area that neither has completed. Also, for a given environment, different initial positions of robots may lead to different GRDs, since the nature and timing of the cooperation may change.

2.2 CC_{RM} description

To perform cooperative coverage, each robot must first be able to perform coverage by itself. This is the job of CC_{RM}, a sensor-based coverage algorithm for a rectangular robot with only intrinsic contact sensing oper-

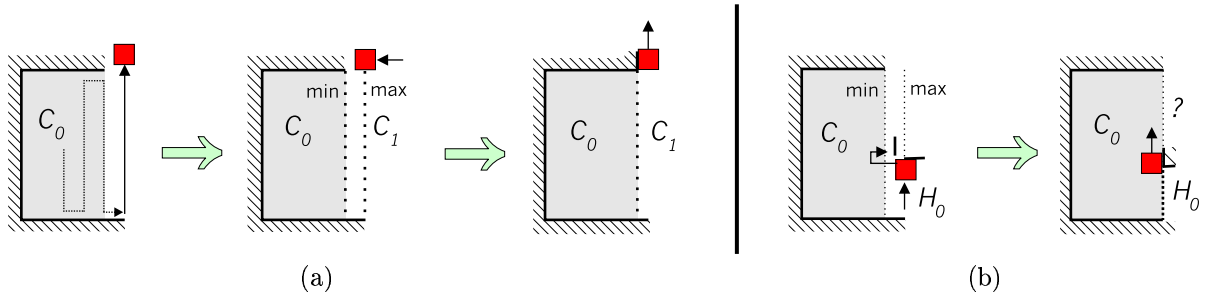

Figure 3: *Some examples of how CC_{RM} discovers and localizes interesting points.*

ating in rectilinear environments. It operates in cycles, with the length of each cycle being a single straight-line trajectory of the robot. At the beginning of each cycle, it examines the structure of **C** around the current position, and uses an ordered list of rules to determine the next trajectory (both direction and distance) required to continue coverage. The rules are structured so as to produce complete coverage. Once a trajectory is submitted to the underlying robot, the robot moves until one of three types of *coverage events* occurs. A coverage event occurs when the maximal distance of the trajectory is achieved, a collision takes place, or contact that is to be maintained with a wall is lost. CC_{RM} then updates **C** given the type of event, chooses a new trajectory, and the cycle repeats.

Under CC_{RM}, each cell C_i is described by its minimum known and maximum potential extents, C_{i_n} and C_{i_x} respectively. Associated with each edge of the cell is a list of *intervals*, each of which is a connected line segment describing the cell's neighbor across that edge. Each interval therefore points to a wall, another cell or a *placeholder*, a line segment denoting the entrance to unexplored area.

To cover each cell, CC_{RM} generates a *seed-sowing* path as shown in the leftmost portion of Fig. 3a, in which the robot travels along paths parallel to its y axis and as far apart as the width of the robot. These continue until an interesting point is detected, such as in the middle portion of Fig. 3a. When this occurs, CC_{RM} updates **C** based on the type of coverage event. In the case of Fig. 3a, a new cell (C_1) is added around p with uncertainty in the boundary between it and the previous cell. A rule then fires based on this uncertainty to localize the interesting point before coverage continues in the new cell. Another case of detection and localization of an interesting point is shown in Fig. 3b, which uses similar rules, although in this case a placeholder H_0 is built rather than a new cell.

Since the first applicable rule determines the next trajectory, rules for interesting points are tested first, followed by the seed-sowing rule, as follows:

1. If p is in a cell, call that cell C_c and continue at rule 2. Otherwise, if p_x is within w (the width of the robot) of a complete cell, go into that cell.
2. If C_c has a side edge with finite uncertainty, move to localize that edge.
3. If C_c has a side edge at a known position that is not completely explored, investigate the closest unexplored point in it.
4. If C_c has unknown floor or ceiling, go in $-y$ or $+y$ respectively.
5. If C_c is not covered from its left edge to its right edge (note that if either edge is unknown, this will always be true), continue seed-sowing.

If none of these rules apply to C_c, then it must be *complete*. A cell is defined as complete when each of its edges are at known location and are spanned by a set of intervals, and its interior has been completely covered with seed-sowing strips. If C_c is complete, the following rules are used, in order:

6. If there is an incomplete cell in **C**, plan a path to it and follow the first step of that path.
7. If C_c has a placeholder neighbor, build a new cell from it and enter the new cell.
8. Choose a placeholder from **C**, plan a path to the cell it adjoins and take the first step of the path.

Path planning is done by implicitly creating a graph from the adjacency relationships of the cells in **C**, and

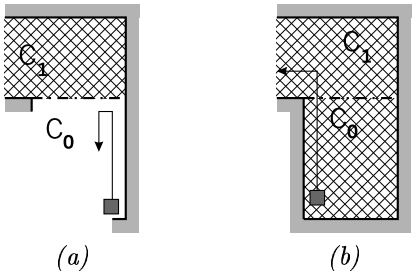

Figure 4: *The effects of an exploration boundary (dash-dot line) when the robot is in (a) an incomplete cell and (b) a complete cell.*

searching that graph for a path to the destination. The search is done depth-first from the destination back to p, and the destination is chosen deterministically, so that even though the plan is regenerated after each trajectory, the planned path will remain the same as the plan is executed. Traversal of each cell is done simply with straight line motions. Finally, if none of these rules fire, there must be no placeholders or incomplete cells, and so $\mathbf{C}$ must be completely covered and its boundary explored, meaning coverage is complete.

Once cooperation is achieved, a cell may have another (complete) cell above or below it, as seen in Fig. 2c. To allow CC_{RM} to continue seed-sowing in such cells, a construct called an *exploration boundary* has been developed. Exploration boundaries are virtual walls placed at the floor and ceiling of each cell obtained from a colleague, and have the property of allowing the robot to pass though them only when the robot is in a complete cell. This allows CC_{RM} to perform seed-sowing in a cell that does not have walls at its floor or ceiling, as shown in Fig. 4a, but does not impede path planning once the cell is complete, as shown in Fig. 4b. It should also be noted that in any case, a completed cell will always have an *attached* floor and ceiling, where an attached cell edge is one that is adjacent to a wall or another complete cell (not a placeholder) at every point.

A second addition made to CC_{RM} that only has impact during cooperation is that a robot can "claim" a placeholder as it builds a cell from it. The robot passes this claim on to the overseers of its colleagues, and so the other robots in the team will not travel to that area to perform coverage. This helps minimize the double covering of area, but also implies trustworthiness of the robots. However, the claiming of area does not effect completeness of DC_R, so it can be implemented only when desired.

2.3 Feature Handler description

The feature handler is quite independent from the other two components in its behavior. Its task is to use the data in $\mathbf{C}$ to generate colleague relationships between robots, however, it does not alter $\mathbf{C}$. Therefore, the specific types of features and algorithms used by the feature handler can change from one system to the next without affecting the rest of DC_R. For example, in the current implementation, the feature handlers look for distances between unlabeled landmarks (beacons) that are common to two robots' maps. However, other feature types such as wall lengths or beacon labels (if present) could also be used, depending on the particular system. The generic behavior of a feature handler is to inform all other robots in the team of the values for all instances of a specified feature type in $\mathbf{C}$. When two robots discover matching features, their feature handlers symmetrically compute the relative transforms between their local coordinate systems using appropriate geometric algorithms. These transforms are then used by the overseer when incorporating data obtained from colleagues.

The other mandatory job of the feature handler is to transfer data to all colleagues at the correct times. When a colleague relationship is formed, all complete cells must be given to the colleague, and when a cell in $\mathbf{C}$ first becomes complete, that cell must be reported to all colleagues. This ensures that the information available to each robot is consistent, which in turn maintains the attached edge property of cells described above.

2.4 Overseer description

The overseer has the task of incorporating all data from colleagues into $\mathbf{C}$, a job complicated by the requirement that $\mathbf{C}$ must remain admissible to CC_{RM}. The addition of an incoming cell C_{new} to $\mathbf{C}$ is done in two stages. In the first stage, zero or more new cells are added to $\mathbf{C}$ to account for the area of C_{new}. Then, for each added cell, its intervals are assigned to walls, existing cells or newly created placeholders. An example of the action of the overseer is shown in Fig. 5.

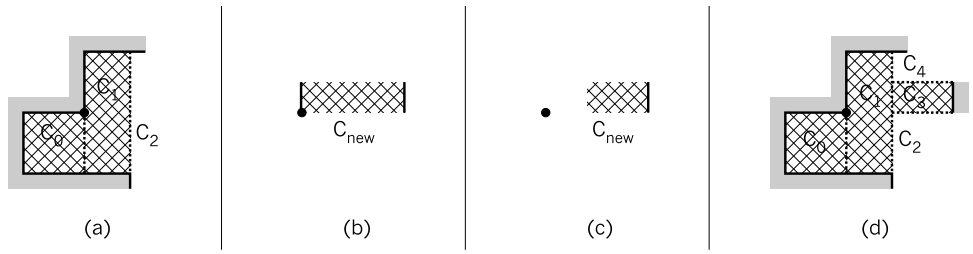

Figure 5: *An example of adding new area by the overseer, in which the initial cell decomposition is depicted in Fig. 5a and the incoming cell C_{new} in Fig. 5b. The dot in each section of the figure represents a common real-world point.*

The cell C_{new} arrives described in the coordinate system of the sending robot, and so it is first transformed into the local coordinate system using the transform provided by the feature handler. Also at this time, all intervals in C_{new} that do not point to walls are modified to point to "unidentified free space" rather than a specific cell or placeholder, since any such neighbor information in C_{new} is meaningful only to the robot that sent it.

To determine the area of cells to be added to $\mathbf{C}$, $\mathbf{C_{com}}$ is defined as the set of all complete cells in $\mathbf{C}$, and $\mathbf{C_{inc}} = \mathbf{C} - \mathbf{C_{com}}$. For the example in Fig. 5, $\mathbf{C_{com}} = \{C_0, C_1\}$ and $\mathbf{C_{inc}} = \{C_2\}$. C_{new} is then intersected with each cell in $\mathbf{C_{com}}$ as follows:

- $\forall C_i \in \mathbf{C_{com}}$:
  - If $C_i \cap C_{new} = \emptyset$, do nothing.
  - If C_{new} is wider (larger in x) than C_i:
    - If $C_{new,right} > C_{i,right}$, make a copy of C_{new} called C_x, set $C_{x,left} = C_{i,right}$, and call the overseer with C_x.
    - Similarly (note no else here) for $C_{new,left} < C_{i,left}$.
    - If C_{new} is also taller than C_i, make a copy of C_{new} called C_x, set $C_{x,left} = C_{i,left}$ and $C_{x,right} = C_{i,right}$, then call the overseer with C_x.
  - Else if C_{new} is taller than C_i, perform similar tests on top and bottom (no third test).

If C_{new} (or its descendants) survive this process (such as the area shown in Fig. 5c, added to $\mathbf{C}$ as C_3 in Fig. 5d), it will consist only of area new to $\mathbf{C}$, i.e. $C_{new} \cap \mathbf{C_{com}} = \emptyset$. In addition, whether or not the area has been divided, each cell will still have at least two attached edges (either walls or other complete cells, possibly also provided by the sender of C_{new}). Each new cell is then intersected with every cell $C_i \in \mathbf{C_{inc}}$. This intersection process is designed to retain the complete C_{new} and eliminate any overlap with incomplete cells (shrinking or deleting the incomplete cells as necessary). Since an incomplete cell must be attached on its floor and ceiling, C_{new} cannot be taller than any $C_i \in \mathbf{C_{inc}}$. The intersection is therefore performed as follows (note that it is not recursive, since each C_{new} is now a fixed size):

- $\forall C_i \in \mathbf{C_{inc}}$:
  - If $C_{i_x} \cap C_{new} = \emptyset$, do nothing.
  - If $C_{i_n} \cap C_{new} = \emptyset$, reduce C_{i_x} so that it does not overlap C_{new}, skip to next C_i.
  - If $C_{i_n} \cup C_{new} = C_{new}$, replace C_i with C_{new}, skip to next C_i.
  - If C_{new} is the same height as C_i, there must be a partial overlap in the x direction, so reduce C_{i_n} (on either its left or right as appropriate) to abut C_{new}.
  - Otherwise, there must be partial overlap in the y direction:
    - If $C_{new,ceil} < C_{i,ceil}$, replace C_i with a cell C_x, set $C_{x,floor} = C_{new,ceil}$ and keep only placeholders attached to C_x, and create an interval in C_x to point to C_{new}.
    - Similarly (again no else) for $C_{new,floor} > C_{i,floor}$.

In the example, this intersection process results in the decomposition shown in Fig. 5d. Finally, each unassigned interval i in C_{new} is given the correct neighbor(s). This is done by determining which cell's maximum extent (if any) is across from the two ends of i (these cells are denoted C_{top} and C_{bot}). i is then assigned as follows:

- If $C_{top} = C_{bot} = \emptyset$, build a new placeholder H_n equal in size to i and set i's neighbor to H_n.
- If $C_{top} = C_{bot} \neq \emptyset$:
  - If C_{top} is complete, set i's neighbor to C_{top}. Also, find the interval in C_{top} that corresponds to i and connect it to C_{new}.
  - If C_{top} is incomplete and i is horizontal, split i, connect it to C_{top} for $i \cap C_{top_n}$ and build a placeholder for $i \cap C_{top_x}$.
  - If C_{top} is incomplete and i is vertical, connect i to C_{top} over the y extent of C_{top_n} as long as C_{top_n} is within one robot width of i, build a placeholder otherwise. If $C_{top,n}$ adjoins C_{new}, find the corresponding interval in C_{top} and connect it to C_{new}.
- If $C_{top} \neq C_{bot}$, this should only occur if i is horizontal and C_{top} and C_{bot} are each either an incomplete cell or $\emptyset$. In this case, connect i where adjacent to C_{top_n}, C_{bot_n} to those cells, and build placeholders for the remainder of i.

3 Proof

To prove that DC_R leads to complete coverage of any finite rectilinear environment by any number of robots (in the absence of interrobot collisions), it is first necessary to show the completeness of CC_{RM}, since DC_R run by a single robot is exactly CC_{RM}. We then show that any cooperation regardless of its timing and nature does not interfere with the progress of CC_{RM}. These statements, combined with the reactive nature of CC_{RM} (and therefore the decoupled nature of coverage and cooperation under DC_R), imply that DC_R is complete.

Proposition 1 *CC_{RM} continues coverage to completion in a finite rectilinear environment in the absence of cooperation that alters the robot's current cell.*

Completeness of CC_{RM} is shown through the construction and analysis of a finite state machine (FSM) which represents all possible behavior of CC_{RM}. To construct the FSM, an equivalence relation is defined over all possible cell decompositions $\mathbf{C}$ and robot positions p such that any two $(\mathbf{C}, p)$ pairs that cause the same rule to be applied in the same way are considered equivalent. The resulting equivalence classes define the states of the FSM. All possible motions of the robot under CC_{RM} (without cooperation) are then enumerated, starting from the initial condition of an empty map. Each motion has from one to three (uncontrollable) outcomes, each of which is a different type of coverage event and is represented by a transition in the FSM. Completeness is then demonstrated by showing that there are no terminal states other than complete coverage, and no cycles that do not result in a monotonically increasing measure of progress toward coverage. In addition, since states of the FSM are uniquely determined by the robot's current cell C_c, the FSM is valid for any valid GRD that includes C_c (no matter how it was constructed). A simplified version of the FSM, in which some nodes consist of several states and transitions, is shown in Fig. 6. Note that all loops in Fig. 6 contain a cell completion event, which indicates progress toward complete coverage. While the details of the FSM are beyond the scope of this presentation, a brief description of each node may give some insight to the structure of the FSM as well as CC_{RM} itself.

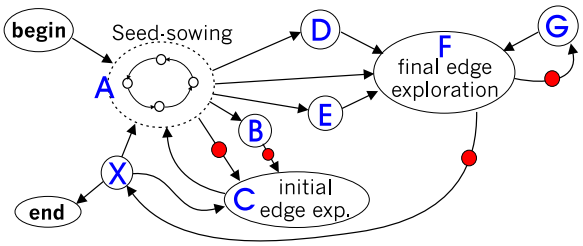

Figure 6: *A simplified version of the FSM representation of CC_{RM}. Contents of the nodes are described in the context of the proof. Gray dots represent the completion of a cell.*

Node A contains the states that describe seed-sowing. In the absence of another incomplete cell overlapping the current cell, seed-sowing continues, cycling through the four states in node A, until an interesting point is discovered. This discovery can be made in five different ways, each leading to one of five successor states of node A.

Node B contains a single state, the situation shown in the middle of Fig. 3a. From this state, a single motion completes the previous cell, and the robot is then in a cell with one known and partially explored edge. All states for which the current cell has this structure are contained in node C, in which the robot explores this first side of the cell. Node C contains two cycles

which correspond to the exploration of that edge monotonically up or down ($+y$ or $-y$). As the exploration progresses, the robot may create and extend wall intervals and/or placeholders, but there is never a cause to reverse direction. This exploration may also include attachment of the current cell to neighboring cells, which is done correctly in any valid GRD. Once this exploration is complete, seed-sowing once again begins in this new cell.

Nodes D and E each represent a single state which will always lead to node F. Node D represents a state in which a corner is discovered by losing contact while moving along the floor or ceiling of a cell during seed-sowing. Node E represents the case shown in Fig. 3b, in which a placeholder is built and the final edge of a cell localized. In both of these cases, the cell's edge is then at a known location. Node F in turn applies to all cells where all but one edge are known and explored. It is similar to node C, in that it contains several states that describe the exploration of an edge, and the robot moves monotonically along the edge. There are some internal differences between the nodes, and when the exploration is complete in node F, so is the cell, with one exception. This exception is for the first cell (C_0), for which exploration of its first known edge is also described by node F, leading to node X. This can only happen once, however, so this cycle cannot be repeatedly traversed without completing a cell.

Finally, node X describes the state where the robot's current cell is complete, as well as when the first edge of C_0 has been explored. In the latter case, since C_0 is not yet complete, CC_{RM} returns to node A to perform seed-sowing toward the other side of C_0. Otherwise, there are three possibilities. If there is an incomplete cell in $\mathbf{C}$, the robot will enter it and restart seed-sowing in node A. Otherwise, if a placeholder exists, CC_{RM} will build a cell from the placeholder and begin to explore the cell's near edge as described by node C. Finally, if and only if node X is reached and no incomplete cells or placeholders exist, coverage is complete and DC_R terminates.

Proposition 2 *The action of the overseer always leaves* $(\mathbf{C}, p)$ *in the domain of* CC_{RM}.

This proposition has both global and local (to p) implications. Globally, the overseer must always produce a valid GRD, since this is assumed in Proposition 1. In addition, the local construction of $\mathbf{C}$ must be such that $\mathbf{C}$ and p form a state which is represented in the FSM described above. Global correctness is demonstrated by showing that the area of any cells added to $\mathbf{C}$ is correct and that all mutual connections between cells are constructed correctly. Local correctness is then shown via an enumeration of the possible effects on the robot's current cell by the overseer.

Proof that added area is correct relies on the fact that all complete cells in a GRD are supersets of SID cells (including the cell obtained by the overseer C_{new}). Therefore, $\forall C_i \in \mathbf{C_{com}}, (C_{new} - C_i)$ is also a superset of SID cells. To confirm that the area added by the overseer from C_{new} is in fact $(C_{new} - C_i)$, C_{new} is written as $C_l \cup C_m \cup C_r$, where C_l is the area to the left of C_i, C_r the area to the right of C_i, and C_m the remainder of C_{new}. C_l and C_r will be fed back to the overseer if non-null, at which point they will remain unchanged relative to C_i. If C_m is larger than C_i, it will also be given to the overseer, but will be subject to the vertical intersection test, which in turn sends $C_m - C_i$ to the overseer. Otherwise, $C_m \subset C_i$, or equivalently $C_m - C_i = \emptyset$. In either case, the total area added based on C_{new} and C_i is $(C_l \cup C_r \cup [C_m - C_i]) = (C_{new} - C_i)$. For incomplete cells $\{C_i : C_i \in \mathbf{C_{inc}}, C_i \cap C_{new} \neq \emptyset\}$, it must then be shown that after C_{new} is added, all known edges of C_i lie on edges of the SID. Since in all cases, edges of C_i that are moved will be coincident with edges of C_{new}, which is a valid GRD cell, this condition is also satisfied.

To show that each added cell is correctly connected to its neighbors, it must be shown that determining the neighbors at each end of an interval (as is done by the overseer) is sufficient to determine its overall disposition. This is in turn true if no interval is adjacent to more than two cells, and no cell lies adjacent to an interval without reaching one end of it. Proofs of these statements rely on the property that each complete cell in a GRD will always have two attached opposing edges.

To show that no cell can lie in the middle of an interval i, assume that such a cell exists (call it C_x). By definition, C_x must not have a neighbor on the side that attaches to i, otherwise that neighbor would be present instead of C_{new}. C_x must therefore have neighbors on its sides perpendicular to i. However, if these neighbors are walls, i will end at the edges of C_x, which contradicts the original assumption. Otherwise, the two neighbors must themselves have neighbors in the

Cell relation	description	$p \in C_{new}$	$p \notin C_{new}$
$C_{new} \cap C_{c_x} = \emptyset$	no overlap	—	no effect[†]
$C_{new} \cap C_{c_n} = \emptyset, C_{new} \cap C_{c_x} \neq \emptyset$	overlap maxsize only	case 1 in text	Continue in C_c
$C_{new} \cap C_{c_n} = C_{c_n}$	cell subsumed	in node X	case 2 in text
$C_{new} \cap C_{c_n} \subset_y C_{c_n}$	top/bottom replaced	in node X	Continue in small C_c
	middle replaced	as above, but see case 3 in text	

[†]It is possible for the intervals on an edge of C_c to be modified by the addition of an adjacent cell, but this will not change the state of CC_{RM} except perhaps to complete C_c.

Table 1: *Effects of the overseer on the robot's current cell C_c.*

directions perpendicular to i. Eventually this chain of cells must adjoin a wall, at which point i's neighbor will be a cell, not null as assumed.

Proving that an interval i will not adjoin more than two cells is also shown by contradiction. First, assume i is a vertical interval in C_{new} adjacent to two existing cells. The two cells (C_a and C_b, complete or incomplete) must therefore be unattached on the side facing C_{new}. C_a and C_b must therefore have attached floors and ceilings (including their mutual edge), and both must therefore have been created by a robot with the same sweep direction. But this is not possible, as one would have to be created first (even if both came from different robots), and would therefore have to extend to a true wall, not just to the other cell. Two cells like this C_a and C_b therefore cannot exist, and so a vertical interval cannot have more than one neighbor. For a horizontal interval, the argument works exactly the same way for complete cells. However, in this case it is possible to have at most two incomplete cells adjacent to i if and only if they are C_0 and C_1.

Finally, to show that the action of the overseer does not leave the robot in a position from which coverage cannot continue, the possible actions of the overseer in the neighborhood of p must be investigated. If the robot is in a complete cell at the time of cooperation, this cell will not be changed, and so the state of CC_{RM} is likewise unchanged. Otherwise, the robot's current cell C_c is incomplete, and so the possible intersections of C_{new} and C_c must be enumerated. However, C_{new} must be no taller than C_{c_n}, since any known neighbors above and below C_c must be walls or complete cells. The enumeration of all cases of intersection of C_c and C_{new} is therefore as shown in Table 1, with the resultant state of CC_{RM} also dependent on whether p is within C_{new}.

Three of the results from this intersection are nontrivial, and are presented here in more detail. Case 1 is shown in Fig. 7a, and is handled by connecting the interval in C_{new} to C_c even though the minimum extents of the two cells do not adjoin. This connection is prescribed in Sec. 2.4. This puts CC_{RM} in node X, but allows the robot to reenter C_c, as it is required to do, since C_c is still incomplete. CC_{RM} then resumes seed-sowing, since C_c still does not extend to C_{new}. Case 2, shown in Fig. 7b, is handled by rule 1 of CC_{RM} (this is in fact the only case in which this rule produces motion). Now in node X, a move in the x direction will always succeed, since C_{new} must be as tall as C_c, and if there was a wall between p and C_c, it would have been discovered already. The robot then continues from this complete cell. Finally, case 3 in Table 1 is one in which multiple incomplete cells are created, such as in Fig. 5d, which could potentially cause problems for seed-sowing. However, these new cells will always share a known edge, and coverage can only continue away from that edge. Also, the potential failure of seed-sowing under CC_{RM} can only occur when moving through the known but unexplored edge of an incomplete cell. Therefore, the robot will always complete one of the new cells (even with successors) and return to the other without triggering a failure of CC_{RM}.

Proposition 3 *Propositions 1 and 2 are sufficient to prove completeness of DC_R.*

Proposition 2 ensures that regardless of the input to the overseer, CC_{RM} will always find itself in the FSM described in Proposition 1, from which it will continue to perform coverage, making monotonic progress. Also, since the overseer can only increase the area spanned by $\mathbf{C_{com}}$ (or leave it unchanged), the act of coopera-

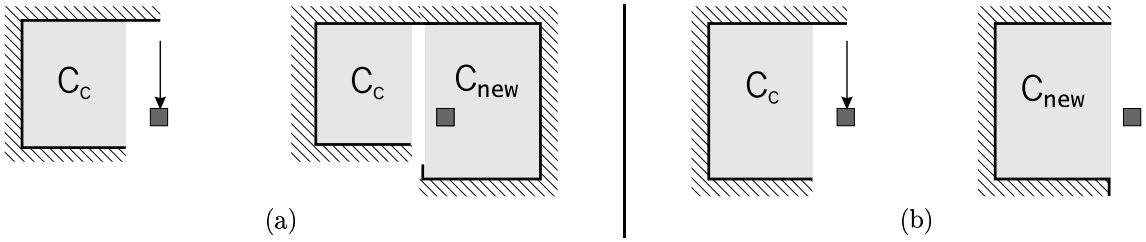

Figure 7: *(a) A case where the robot can't get to an incomplete cell and (b) a case where the robot is left outside* **C**.

tion also describes monotonic progress toward complete coverage.

4 Implementation / Example

DC_R has been implemented in simulation, and a series of screen shots of a single run are shown in Fig. 8. The simulation gives an overall view of the environment as coverage progresses, and also displays a view (not shown here) of the cell decomposition internal to each robot. In the simple example shown here, the two robots R_0 and R_1 performed coverage independently until a time just before Fig. 8b was taken, at which point their feature handlers decided that sufficient information was present in their maps to determine their relative position. Note that at this point, R_0 is no longer performing seed-sowing over the full width of the environment. By the time Fig. 8c was taken, each robot was working on a different area of the environment, after which they ended up in the same area. R_1 then claimed the area at the lower left before R_0 could, and so R_0 simply waited in a "safe" place for R_1 to finish. Finally, when coverage is complete in Fig. 8d, note that only about half of the area has been visited by both robots. While clearly not optimal with respect to time or total distance for the pair, it does show that each robot spends less time covering than it would without cooperation.

In addition to the simulation, CC_{RM} has been successfully implemented in the minifactory environment on a single courier, and the relationship of the algorithm to the underlying robot control will remain the same in DC_R. However, some extensions to DC_R will ease its transition into a real-world robot system. For example, the simulation currently incorporates small non-cumulative position error, which is allowed for in most cases, but not in a rigorous (or even completely correct) manner. Further analysis of the effects of this and other types of uncertainty will make DC_R applicable to a wider variety of robot systems. Also, while the simulation can effect collisions between robots, DC_R currently uses simple methods to attempt to avoid colleagues and make progress. These methods often succeed, but are prone to failure in complex environments, and more intelligent strategies must be developed, preferably ones that allow the completeness proof to be retained with minimal modification. Our eventual goal is the implementation of DC_R on the minifactory hardware to verify its utility in a real-world system.

5 Conclusion

In this paper, an algorithm DC_R has been presented with which a team of independent robots can cooperatively cover their shared environment. It comprises a reactive coverage algorithm CC_{RM} which operates without explicit knowledge of cooperation and two additional components (the feature handler and the overseer) which maintain cooperative relationships with other robots to increase efficiency of the team. This decoupling of coverage from cooperation enabled the completeness proof of DC_R presented in this paper, demonstrating that any team of square robots with intrinsic contact sensing can successfully cover a finite rectilinear environment efficiently.

Acknowledgements

The authors would like to thank Howie Choset and Ercan Acar for helpful discussions about coverage. This research was funded in part by NSF grant DMI-9527190. Zack Butler was supported in part by an NSF Graduate Research Fellowship.

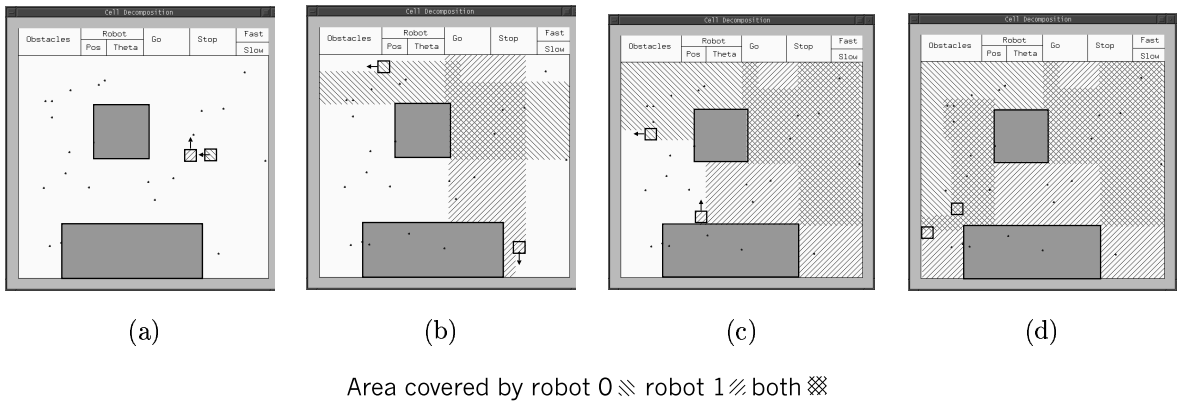

Figure 8: *Screenshots of a two-robot coverage run in progress: (a) Initial positions (b) Just after colleague relationship is formed (c) Each robot exploring a different region (d) Coverage is complete.*

References

[1] Y. S. Suh and K. Lee, "NC milling tool path generation for arbitrary pockets defined by sculptured surfaces," *Computer Aided Design*, vol. 22, no. 5, pp. 273–284, 1990.

[2] V. Lumelsky, S. Mukhopadhyay, and K. Sun, "Dynamic path planning in sensor-based terrain acquisition," *IEEE Trans. on Robotics and Automation*, vol. 6, no. 4, pp. 462–472, 1990.

[3] S. Hert, S. Tiwari, and V. Lumelsky, "A terrain covering algorithm for an AUV," *Autonomous Robots*, vol. 3, pp. 91–119, 1996.

[4] E. Acar and H. Choset, "Critical point sensing in unknown environments for mapping," in *IEEE Int'l Conf. on Robotics and Automation*, 2000.

[5] H. Choset and P. Pignon, "Coverage path planning: The boustrophedon decomposition," in *Intl. Conf. on Field and Service Robotics*, 1997.

[6] Z. J. Butler, A. A. Rizzi, and R. L. Hollis, "Contact sensor-based coverage of rectilinear environments," in *Proc. of IEEE Int'l Symposium on Intelligent Control*, Sept. 1999.

[7] A. A. Rizzi, J. Gowdy, and R. L. Hollis, "Agile assembly architecture: An agent-based approach to modular precision assembly systems," in *Proc. of IEEE Int'l. Conf. on Robotics and Automation*, pp. 1511–1516, April 1997.

[8] B. R. Donald, J. Jennings, and D. Rus, "Information invariants for distributed manipulation," *International Journal of Robotics Research*, vol. 16, no. 5, pp. 673–702, 1997.

[9] A. Drogoul and J. Ferber, "From Tom Thumb to the dockers: Some experiments with foraging robots," in *From Animals to Animats II*, pp. 451–460, MIT Press, 1993.

[10] P. Stone and M. Veloso, "Task decomposition, dynamic role assignment and low-bandwidth communication for real-time strategic teamwork," *Artificial Intelligence*, vol. 110, pp. 241–273, June 1999.

[11] D. W. Gage, "Randomized search strategies with imperfect sensors," in *Mobile Robots VIII*, pp. 270–279, 1993.

[12] N. Rao, V. Protopopescu, and N. Manickam, "Cooperative terrain model acquisition by a team of two or three point-robots," in *Proc. of IEEE Int'l. Conf. on Robotics and Automation*, pp. 1427–1433, April 1996.

[13] I. Rekleitis, G. Dudek, and E. Milios, "Multi-robot exploration of an unknown environment, efficiently reducing the odometry error," in *Int'l Joint Conf. in Artificial Intelligence*, (Nagoya, Japan), pp. 1340–1345, August 1997.