(67686) Mathematical Foundations of AI May 21, 2008

Lecture 2

Lecturer: Ariel D. Procaccia *Scribe: Reshef Meir, Yoni Peleg, Nir Pochter*

1 Robotic Coverage - Introduction

There are many real life applications that require a robot, or a group of robots, to cover an area (meaning to visit it all). For example, a vacuum cleaning robot that needs to clean an entire room, intrusion detection, mine cleaning and search-and-rescue missions. In this lesson we will formally define the single-robot coverage problem and the multi-robot coverage problem, present and prove some of the properties of algorithms that solve these problems.

1.1 Settings

The environment that the robots operate in is represented as a 2D grid of large square cells, that can be either completely blocked or completely unblocked. Each of such large cell is composed of four small cells. All of the robots are of the size of a small cell, and can be located in one of these cells at any time, if it is unblocked. A robot that is located in an unblocked cell can move to any adjacent cell, if it is unblocked (meaning the robot can move up,down, left and right, but not in diagonal moves). Such a move is assumed to take unit time (the time is equal for all robots). It is possible for more than one robot can occupy the same cell at the same time.

1.2 Single-Robot Coverage - Simple Algorithm

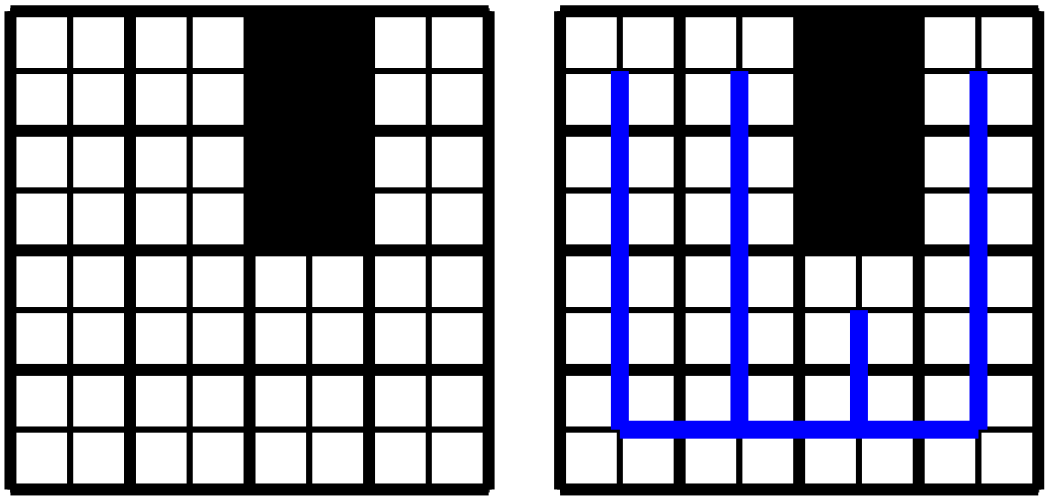
In this problem the environment is represented as described in Section 1.1. There is one robot, with a known starting location. The goal is to find a coverage. We will now describe the Spanning Tree Coverage algorithm:

1. Define a graph G whose vertices are the centers of all the large cells, and whose edges connect adjacent unblocked large cells.
2. Find a spanning tree for the graph.
3. The robot will now circumnavigate the spanning tree

Theorem 1 *The STC algorithm covers every small cell that is accessible from the starting cell S.*

Proof Since the algorithm circumnavigates the spanning tree, it passes at each small cell that touches the spanning tree. We assumed that every unblocked small cell is part of an unblocked large cell. All centers of unblocked large cells are vertices in the graph, and therefore, from the way the MST is built, the robot has to pass through any small cell. ■

Theorem 2 *The STC algorithm is optimal.*



(a) A grid representation of the environment

(b) The grid representation with a spanning tree drawn in it

Figure 1: Example for the STC algorithm

Proof We showed that the robot who follows the algorithm passes through each cell at least once. We will now show that the robot does not pass in any cell twice. Since the robot follows the tree in a clockwise direction, in order to pass in the same small cell twice before it returns to the original cell, there must be a small cell that is adjacent to two parallel edges of the MST. From the construction of the MST each two edges that don't start from the same large cell are at least two small cells apart, which makes it impossible. Therefore, the algorithm cannot visit the same small cell more than once, which makes it optimal. ■

1.3 Multi-Robot Coverage - Simple Algorithm

We will now look at the multi-robot coverage problem: we are given the location of r robots. At each time step each of the robot makes one move. The coverage time is the time that passes until at least one robot visited each small cell. To solve the multi-robot coverage problem, STC was extended to MSTC — multi robot spanning tree coverage. The algorithm is as follows:

1. Define a graph G whose vertices are all the large cells, and whose edges connect adjacent unblocked large cells.
2. Find a spanning tree for the graph.
3. Each robot will move clockwise on the path that circles around the spanning tree, until it reaches the starting location of the next robot.

Note that the coverage time is the maximum out of the time that it takes for each robot to walk

Claim: The efficiency of the algorithm is highly dependent of the specific spanning tree we choose. For example, see figure 1.3.

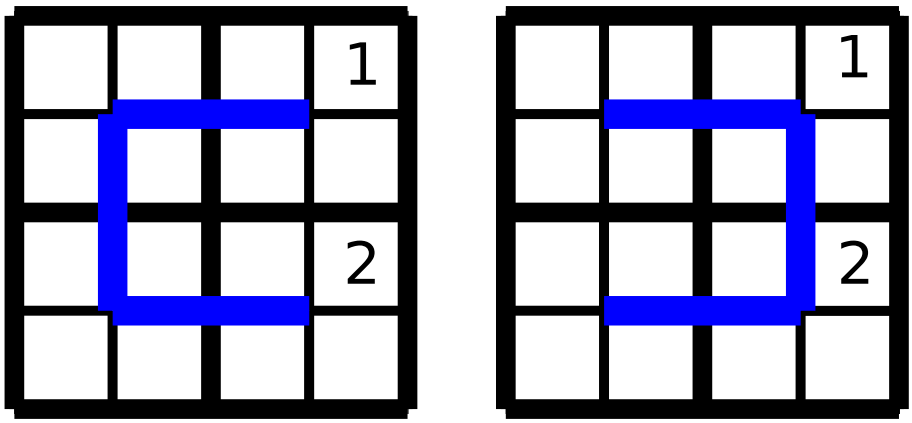


Figure 2: An example for the difference of MSTC performance with different spanning trees. The tree to the left would yield coverage time of 10, while the tree to the right would yield a coverage time of 14.

This leads to the following open problem:
Open problem: Prove/disprove: finding the best minimum spanning tree for MSTC is NP-complete.

2 The Hardness Of The Mobile Robots Terrain Coverage Problem

In this part of the lecture we'll show that the **Terrain Coverage** problem is an NP-complete problem. In other words, we'll show that (assuming that $P \neq NP$) the terrain coverage problem can't be solved in poly-time. We'll prove that the Terrain Coverage problem is NP complete using a reduction from another famous NP complete problem - the *3-partition* problem.

We'll start by giving a quick reminder about some important topics from the computation theory course. Then we'll define the 3-partition problem and the terrain coverage problem (as a decision problem). Finally, we'll show and prove the reduction.

2.1 A Quick Reminder About Reductions

The following section gives a very quick reminder about topics from the *computation theory course*. For more details we refer the students to the course website.

Definition 3 *We'll say that a language L ($L \subseteq \{0,1\}^*$) is in the set P ($L \in P$) if and only if there exists a **deterministic** polynomial Turing machine A such that for each $x \in \{0,1\}^*$, A accepts x if $x \in L$ and rejects it otherwise.*

Definition 4 *We'll say that a language L ($L \subseteq \{0,1\}^*$) is in the set NP ($L \in NP$) if and only if there exists a* ***non-deterministic*** *polynomial Turing machine A such that for each $x \in \{0,1\}^*$, A accepts x if $x \in L$ and rejects it otherwise.*

An equivalent definition: $L \in NP$ if and only if there exists a deterministic polynomial Turing machine A such that for each $x \in L$ there exists "a proof" $y \in \{0,1\}^*$ such that A accepts (x,y) and for each $x \notin L$,$y \in \{0,1\}^*$, A rejects (x,y).

Definition 5 *We'll say that there exists a* Karp Reduction *from the language $L_1 \subseteq \{0,1\}^*$ to the language $L_2 \subseteq \{0,1\}^*$ if and only if there exists a function f that can be calculated in polynomial time such that $x \in L_1$ if and only if $f(x) \in L_2$.*

Intuitively, if there exists a reduction from L_1 to L_2 then the language L_2 is harder than L_1 (because given a Turing machine that accepts L_2 we can use it as a black box in a Turing machine that accepts L_1).

Definition 6 *We'll say that a language L ($L \subseteq \{0,1\}^*$) is NPC (NP Complete) if and only if there exists a reduction from each $L^* \in NP$ to L.*

It's a common belief that $P \neq NP$. Assuming that $P \neq NP$ we'll get that the NP complete problems can't be solved in polynomial time.
In order to show that the language L is in NPC, it's enough to prove that there exists a Karp reduction from a language $L_1 \in NPC$ to L.

2.2 Description Of The Reduction

We wish to show that the Terrain Coverage Problem is an NP-Complete problem. First we should note that above we have considered the Terrain Coverage Problem as an **optimization** problem — the problem was to find the minimum time it'll take the robots to cover the grid. When talking about reductions we must consider a **decision** problem. The following definition will formally define the Terrain Coverage Problem as a decision problem:

Definition 7 *The* Terrain Coverage *decision problem is the language L_{TC} that is defined as follow: We're given a word $w \in \{0,1\}^*$ that describes:*

- *A $m \times n$ grid, where each square might be either empty or blocked.*
- *Each square in the grid is divided into 4 sub-squares.*
- *r robots are located in a given initial position on the grid (each robot is located in a sub-square, it's possible that more than one robot will occupy the same square at a given time).*
- *An integer K.*

The word w is in the language L_{TC} if and only if it's possible to cover the $m \times n$ board in at most K turns. The coverage is done as in the Terrain Coverage optimization problem (the robots may move up,down,left,right from a sub-square to another sub-square of an empty square, in each turn each robot can perform at most 1 move). We'll say that the board has been covered if each sub-square of an empty square in the board was visited by at least one robot, ***and*** *if the robots are back to their original position.*

Note that beside turning the optimization problem into a decision problem, we also add a new constraint: the robots must be back to their original location at the end of the coverage. This constraint simplifies the reduction. Proving that the problem is hard (in NPC) without this constraint is currently an open problem.

So we have formally defined the Terrain Coverage problem as a decision problem. Now we'll formally define the *3-partition* problem which is a known NP-Complete problem we'll use for the reduction:

Definition 8 *The language* $3Pr$ *(*3-partition*) is defined as follow:*
Let $w \in \{0,1\}^*$ *be a word that represents a set* S *containing* $3n$ *integers* $S = \{a_1, a_2, ..., a_{3n}\}$. *The sum of the elements in* S *is* $B \cdot n$ $(\sum_{i=1}^{3n} a_i = B \cdot n)$.
$w \in 3Pr$ *if and only if the elements in* S *can be divided into* n *disjoint sets* $(S_1, S_2, ..., S_n)$ *such that the sum of the elements in each set is equal to* B.

The 3-partition problem is very similar to another very famous NP-Complete problem – the *partition* problem:

Definition 9 *The language* Pr *(*partition*) is defined as follow:*
Let $w \in \{0,1\}^*$ *be a word that represents a set* S *that contains* n *integers. The sum of the elements in* S *is* B. $w \in Pr$ *if and only if the elements in* S *can be divided into* 2 *disjoint sets such that the sum of the elements in each set is equal.*

There is a known solution for the *partition* problem that is based on dynamic programming. If B is polynomial in n, the dynamic programming time complexity is polynomial in the size of the input. Unfortunately, elements of S can be exponential in the number of bits that represent them. In that case the solution complexity is exponential (in the size of the input).

Unlike the partition problem, the **3-partition** problem is a *strongly* NP-Complete problem. The problem will remain hard even if B is polynomial in n.

The name **3**-partition comes from the fact that if each element in S is between $\frac{B}{4}$ to $\frac{B}{2}$ then each subset S_i will contain exactly 3 elements.

We'll use the $3Pr$ language for the reduction since the fact that it remains hard problem even when B is polynomial in n will simplifies the reduction. Nevertheless it's important to emphasize that it's possible to construct a reduction from any other NP-Complete language (including Pr).

It's time to describe our Karp reduction:

Definition 10 *The function* F *(*$F : \{0,1\}^* \to \{0,1\}^*$*) is defined as follow:*
Assume that the input to the function is a word $w \in \{0,1\}^*$ *that represents a set* S *of* $3n$ *integers that sums up to* $B * n$ $(S = \{a_1, a_2, ..., a_{3n}\})$. *Let* $a = max\{a_1, a_2, ..., a_{3n}\}$. $F(w)$ *will present a* $3n \times 12n * a + 1$ *grid such:*

- *All the squares in the* $6n * a + 1$ *column are empty.*
- *For each odd* i $(1 \leq i \leq 3n)$ *the* $6n * a_i$ *squares in the* i*th row right to the* $6n * a + 1$ *column are empty.*
- *For each even* i $(1 \leq i \leq 3n)$ *the* $6n * a_i$ *squares in the* i*th row left to the* $6n * a + 1$ *column are empty.*

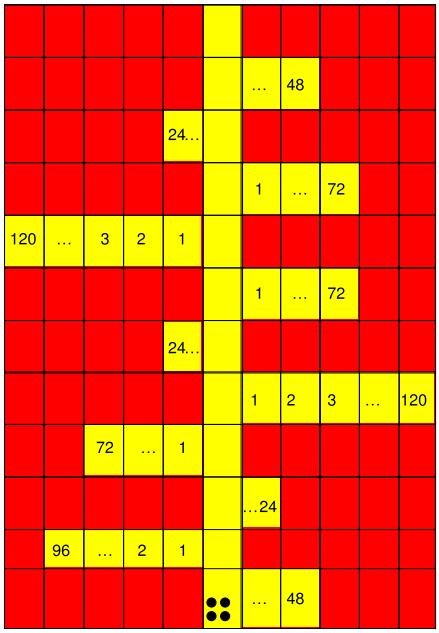


Figure 3: Reduction Example

- *The rest of the squares in the grid are blocked.*
- *n robots will be located at the bottom-left sub-square of the square in the 1st row,* $6n*a+1$ *column (W.L.O.G. we'll count the rows from the bottom up).*

In addition, $F(W)$ *will contain the number* $24B \cdot n + 12n$ *(the number of turns to cover the area).*

Figure 2 demonstrates the reduction for the set $S = \{2, 1, 3, 5, 3, 1, 5, 3, 1, 4, 2\}$[1]. The figure illustrates the 12×241 grid that is the result of applying F on S. The yellow squares are empty squares, the red squares are blocked. The 4 robots are located at the $(1, 121)$ square. Due to the size of the grid we could not give an exact drawing of it. Instead, only the relative lengths of the passageways are given.

2.3 Proving The Reduction Correctness

In order to prove the correctness of the reduction, we need to show that F fulfills all the demands of the Karp reduction.

2.3.1 Polynomial Function

First we'll show that F can be calculated in polynomial time. Again we'll assume that the set S contains $3n$ elements (integers) and we'll mark with a the maximum integer. So the size of the input is $O(3n*log(a))$ bits. The size of the output of the function will be $O(3n(12n*a+1))$ bits.

[1] We abuse the notation of **set** – S may hold the same integer more than once, using the notation of multi-set will be more accurate here.

The function makes one pass on the input, calculates the output in $O(1)$ and writes the output. So if $a = Poly(n)$ clearly the function is polynomial as required. If, however, a is exponential in n (for example, $a = 2^n$) then for polynomial input ($O(n^2)$ in our example) the function will generate exponential output ($O(2^n)$ in our example). So in order to use the Karp reduction, we have to restrict ourselves to the words in $3Pr$ that represents 3-partition problems in which a is polynomial in n. Fortunately, we can do it since 3-partition is strongly NP-Complete, so the problem will remain hard even in the last case. We'll omit the precise details of the last argument, students that are interested in more information about the subject are referred to the complexity course.

2.3.2 First Direction

Assume that $x \in 3Pr$, i.e., x represents a 3-partition problem such that a solution for the problem exists. As usual we'll denote with $S = \{a_1, a_2, ..., a_{3n}\}$ the original set and we'll denote with $S_1, S_2, ..., S_n$ the partition of S into n disjoint subsets such that the sum of the integers in each subset is B.

Let us consider $F(x)$. We'll show that there is a cover of the resulting grid in time at most $24B \cdot n + 12n$. We'll get the cover by ordering the ith robot ($1 \leq i \leq n$) to walk through the central corridor (the $a + 1$ row) and to enter the jth passageway if and only if $a_j \in S_i$.[2]

All that left for us to show is that the cover time is indeed smaller then $12n + 24B \cdot n$. The time that it'll take each of the robots to walk through the central corridor is exactly $4 \cdot 3n$ ($3n$ – the length of the corridor, we multiply it by 4 since each square is divided into 4 sub-squares). The length of the jth passageway is $6n \cdot a_j$ so the time to cover the passageway will be $4 \sum_{a_i \in S_j} 6n \cdot a_i$ and we'll get:

$$4 \cdot 3n + 4 \sum_{a_i \in S_j} 6n \cdot a_i \leq 12n + 24B \cdot n$$

so $F(x) \in L_{TC}$ as required.

2.3.3 Second Direction

Now we're assuming that $F(x) \in L_{TC}$, i.e., the resulting grid can be covered in $24B \cdot n + 12n$ turns. We would like to show that $x \in 3Pr$, i.e., that there exists a partition of the set S such that the sum of the elements in each disjoin subset is B. We'll construct the partition by setting S_i to be the set of all the passageways that the ith robot was the first robot to reach the square in their edge (the empty square that belong to the square that is the most distant from the center).

The idea: we'll use the fact that the robots must come back to their original position. Thus, if a robot i got to the end of the jth passageway, it must get back to the beginning of it. From the restriction on the coverage time we'll get:

$$4 \sum_{a_i \in S_j} 6n \cdot a_i \leq 24B \cdot n$$

[2] Again, we're abusing the notations here since S_i is a multi-set.

and thus:

$$\sum_{a_i \in S_j} a_i \leq B$$

So we manage to find a partition of S such that each subset S_i contains integers that sum up to at most B. In order to solve the partition problem, we need to show that the elements in each S_i sum up to **exactly** B. However, since all the elements in S sums up to $B \cdot n$, if there exists a subset such that $\sum_{a_i \in S_j} a_i < B$ there must also exists a subset such that $\sum_{a_i \in S_k} a_i > B$, which contradicts our previous proof.

So we manage to find a partition of S such that each subset S_i contains integers that sum up to exactly B, and thus $x \in 3Pr$ as required.

3 Approximating the Multi-Robot Cover problem

In this section we will present a new algorithm that always returns a solution for the Multi-robot Cover Problem that is almost optimal, In the sense that the coverage time is no more than 8 times the optimal solution. We will also show that the MSTC algorithm presented in the first section performs very poorly when more than one robot is used.

3.1 Approximation algorithms

Definition 11 *(Minimization problem)*
Denote by $OPT(I)$ the optimal (minimal) solution for Input I. Algorithm $\mathcal{L}$ is ρ-approximating if for any input I, it holds that

$$\mathcal{L}(I) \leq \rho \cdot OPT(I)$$

and $\mathcal{L}$ runs in polynomial time.

Notes

- By comparing the solutions, we are actually comparing their *quality*, as the solution itself may not be a number.
- For a maximization problem, we demand that $\mathcal{L}(I) \geq \frac{1}{\rho} \cdot OPT(I)$.
- If ρ is a constant number that does not depend on I, we say that $\mathcal{L}$ provides a *constant* approximation ratio.

Approximation algorithms are one of the most prominent ways to confront problems that are known (or suspected) to be computationally hard, such as problems that are $\mathcal{NP}$-complete. For some problems (e.g. Vertex Cover) there are simple algorithms that guarantee a constant approximation, sometimes even for a constant that is arbitrarily close to 1 (e.g. the 0-1 Knapsack). For other problems (e.g. Clique, Set Cover) it can be shown that finding a constant approximation, for any constant, is itself an $\mathcal{NP}$-hard problem.

3.2 The Rooted Tree Cover Problem (RTC)

In this problem, we are trying to cover a graph with a set of trees, whose roots are fixed vertices of the graph.

Formally, let $G = (V, E)$ denote an undirected graph with positive integral edge weights $w : E \to \mathbb{N}_+$. Let $R \subseteq V$ denote a set of roots. An *R-Rooted Tree Cover* of a G is a set T of trees $\{T_i\}_{i=1}^k$ such that

- $V = \bigcup_{i=1}^k V(T_i)$, i.e. the trees cover together all the nodes of G.
- each tree T_i has a distinct root in R.

Trees in an R-rooted tree cover may share nodes and edges. In particular, the root of T_i may be in T_j , but the roots of T_j and T_i must be distinct. The weight of each tree T_i is defined by $w(T_i) = \sum_{e \in T_i} w(e)$. The cost of a tree cover T is the weight of the heaviest tree, i.e. $max_{T_i \in T} w(Ti)$.

Given an edge weighted graph G and a set R of roots, the min-max R-rooted tree cover problem is to find a *minimum* cost R-rooted tree cover T of G.

Theorem 12 (Even et al. [1]) *There is a 4-approximating algorithm $\mathcal{A}$ for the RTC problem.*

We will not prove this theorem. The algorithm and proof can be found in [1].

3.3 Multi-robot Forest Cover (MFC)

We will use algorithm $\mathcal{A}$ from the previous section, to provide an approximate solution to the Multi-robot Cover Problem. The MFC algorithm [2] works as follows:

1. Given an instance of the Multi-robot Cover Problem, create an instance (G, R) of RTC, where G is the same graph that was used for MSTC, R is the set of starting locations, and the weight of every edge is 1. Thus the cost of every tree is simply its *size* - the number of edges it contains.
2. Use algorithm $\mathcal{A}$ on the new instance to get an R-Rooted Tree Cover T.
3. Assign to each robot the tree that is rooted to its starting point.
4. Let each robot walk around its tree, just like in STC.

We will now show that the MFC algorithm is 8-approximating for the Multi-robot Cover Problem. First, it clearly halts in polynomial time, since creating the new input is trivial, and we know that $\mathcal{A}$ halts in polynomial time. Let I any input to the original MRC problem, and J the corresponding input to the RTC problem that we created. We use the following definitions to prove the approximation ratio:

- B is the quality (coverage time) of our solution, i.e. $MFC(I)$
- B^* is the quality of the best possible solution, i.e. $OPT(I)$
- B^*_{UL} is the quality of the best possible solution, for covering only the Top-Left subsquare of each square

- C is the quality of the solution we got for the RTC problem, i.e. $\mathcal{A}(J)$
- C^* is the quality of the best possible solution for RTC, i.e. $OPT(J)$

Using 4 inequalities, we will bound B and show that it cannot be more than 8 times B^*. Walking around each tree takes 4 times the size of the tree, plus 4, thus

$$B = 4C + 4 \tag{1}$$

From Theorem 12 we immediately get that

$$C \leq 4C^* \tag{2}$$

Consider the problem of covering only Top-Left subsquares. Suppose that in the optimal solution, each robot j is covering some set of Top-Left subsquares H_j, using some path P_j. Let S_j a spanning tree for H_j (that is, a tree that connect all nodes that contain some subsquare $h \in H_j$). The number of (large) squares that j has to cross in order to visit all of H_j is at least S_j. The real distance that j has to walk (P_j) is actually twice that number, since we count subsquares and not squares. We can thus see that for each j,

$$2|S_j| \leq |P_j|$$

Since all the nodes of H_j are covered by S_j, and $\bigcup_j H_j$ contains all the nodes of G, the forest $\{S_j\}_j$ is also a solution for RTC. Let $C^{\#}$ the quality of this solution. From the definition of solution quality we get that

$$2C^* \leq 2C^{\#} = max_j(2|S_j|) \leq max_j|P_j| = B^*_{UL} \tag{3}$$

Finally, since covering only part of the subsquares cannot take longer than covering all of them,

$$B^*_{UL} \leq B^* \tag{4}$$

Combining the 4 equations, we find that

$$B = 4C + 4 \leq 16C^* + 4 \leq 8B^*_{UL} + 4 \leq 8B^* + 4$$

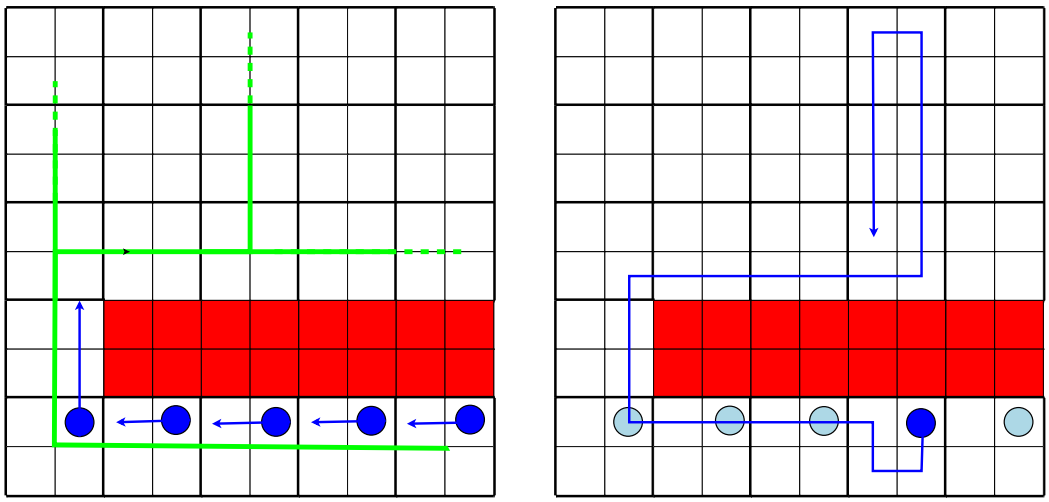
The +4 is ignored, since we are actually interested in the behavior of the algorithm as problems get bigger.

3.4 When does MSTC fail?

We will show a family of inputs of increasing size, for which the approximation achieved by the MSTC algorithm is linear in the number of robots, i.e. for every constant ρ there is an input on which MSTC fails to achieve a ρ-approximation.

Let r the number of robots. We create the following input I_r:

- The grid has $r \times r$ squares
- There is one robot in the top-right subsquare of each square in the first (bottom) row



(a) (b)

Figure 4: A bad instance for MSTC. A partial spanning tree is marked in green.

- All the second row is blocked, apart from the left most square

An illustration of this instance for $r = 5$ is shown in figure 4(a). It is easy to see that the left most robot will do almost all the coverage work by itself, and any other robot will cover exactly 2 subsquares. Thus the total time it takes to cover the entire grid is

$$2r \times 2r - 2(r-1) = \Omega(r^2)$$

On the other hand, there is a way to cover the grid much faster: each robot should cover its own column. The path of one robot is illustrated in figure 4(b). Clearly the right most robot will take the longest time to finish: it has to cover its own column ($4r$ steps) and also to walk around the blocked row ($2(r-1) - 1 + 3 + 2(r-1) = 4r - 2$ steps). The cost of this solution is thus

$$4r + 4r - 2 = O(r)$$

The optimal solution cannot be worse than the solution we described, therefore there exists some fixed number $\alpha > 0$, such that

$$MSTC(I_r) > \alpha \cdot r \cdot OPT(I_r)$$

Which means that for *any* constant ρ, MSTC is not ρ-approximating (just take $r > \frac{\rho}{\alpha}$).

References

[1] G. Even, N. Garg, J. Konemann, R. Ravi, , and A. Sinha. Min-max tree covers of graphs. *Operations Research Letters*, 32:309–315, 2004.

[2] X. Zheng, S. Jain, S. Koenig, and D. Kempe. Multi-robot forest coverage. In *Proceedings of IROS*, pages 3852–3857, 2005.