Project in Intelligent Systems (236754)

Under the supervision of Yotam Elor

Mark & Walk
with the
Ant-Like Robots

By:
Nizar Ashkar, Ebraheem Sabbah

Table of Contents

1) Introduction

In this document we will analyze several runs on various domains using Mark-Ant-Walk algorithm for robust and efficient covering of continuous domains by antlike robots.
We have written a simulator that simulates the movement of robots on given domains according to the algorithm we discuss in this project, using different parameters depending on our purpose, the simulator helps us in outputting result functions that reflect the running time over various areas, we can also gain a graphical documentation of the robot's movement and lots of options depending on our need.
The analyzing is done by studying the several result functions of different domains, we will also present statistics of several runs of the simulation, altering the problem parameters (different domains, different number of robots, etc...) and drawing conclusions on our points of research.

1.1 Objectives & Motivation

Our main goal in this article is to study the influence of the problem parameters on the movement of the robot and the covering time.
We will be looking for methods to finding the most appropriate parameters that reflect approximately a real behavior of the agent(s) in our simulator.
The first problem to solve is representing circles by approximating each covering step of the robot to a real circuit then finding the appropriate radius that reflects this behavior and work with it (this problem and solution will be discussed later elaborately).
After solving the problem discussed above, we are ready to experiment the behavior of the agent(s) on several areas with or without forbidden areas, we will base our research on several parameters; covering time on variable sizes, shapes and number of agents, comparing between them and presenting statistics of several runs.
The last part before we sum our conclusions will include a proposal of experiments we came up with that change the functionality of the algorithm and comparing its results to the original version of the algorithm.

1.2 The Simulator

The simulator was programmed in C++; it was divided into two main modules; Mark-Ant-Walk algorithm methods and the Scheduler which is responsible for manipulating the time unit and the calls of different agents.
It was written with a very comfortable interface for altering the problem parameters and the various research problems we experiment in our project.
We provide a documentation option of the agent(s) steps supported by a simple graphical interface of the robot's movement which was useful in assuring the correctness of our simulation.

2) Preliminaries

2.1 Definitions & Notations

In that section we will define some objects that will serve us with introducing our implementations and presenting our results.

Pheromone Level:

The robots can mark places visited with pheromone marks and sense the level of the pheromone in their local neighborhood.

In case of multiple robots these pheromone marks can be sensed by all robots and provide the only way of (indirect) communication between the robots.

High values of the pheromone level roughly indicate areas that have been visited many times up to the moment and lower values of the pheromone level correspond to a smaller number of robot's visits.

Cover Time:

The number of steps it takes for the agent(s) to cover the given environment; going through every point in the domain and updating the pheromone level in each one, leaving no unmarked points.

Formal Definitions:

- The domain to be covered will be denoted by $\mathbf{\Omega}$.
- Given any two points **a, b**$\in\mathbf{\Omega}$ denote the geodesic distance between **a** and **b** as $\|\mathbf{a}-\mathbf{b}\|$.
- We shall then assume that the robot is able to sense the pheromone level at its current position p and in a closed "geodesic" ring **R(r, 2r, p)** lying between the internal radius r and the external radius 2r around p. **R** is formally defined as follows: $\boldsymbol{R(r1, r2, c)} = \{\boldsymbol{a}, \boldsymbol{b} \in \Omega \mid \boldsymbol{r1} \leq \|\boldsymbol{a}-\boldsymbol{c}\| \leq \boldsymbol{r2}\}$
- The robot is able to set a constant arbitrary pheromone level in the area swept by its effectors, which is, we assume, an **open disk** D(**r, p**) of radius r around its current location p. The formal definition of **D** is as follows: $\boldsymbol{D(r, c)} = \{\boldsymbol{a} \in \Omega \mid \|\boldsymbol{a}-\boldsymbol{c}\| \leq \boldsymbol{r}\}$
- We denote by **σ(a, t)** the pheromone level of point a$\in\Omega$ at time instance t, where t = 0, 1, 2, 3, ...

2.2 Mark-Ant-Walk Algorithm

Initially, we consider the case where no point is marked with the pheromone, thus all σ values are assumed to be equal to zero: $\sigma(a, 0) = 0;\ \forall a \in \Omega$.

```
(1)     Find x: = a point in R(r, 2r, p) with minimal value of σ(x, t)
        (In case of a tie, i.e., when the minimal value achieved at several places - make an arbitrary decision)
        /* note that $\|p - x\| \geq r$ */
(2)     If $\sigma(p) \leq \sigma(x)$ then $\forall u \in D(r, p)$ set $\sigma(u) = \sigma(x) + 1$
        /* we mark open disk of radius r around current location */
(3)     t := t + 1
(4)     move to x.
```

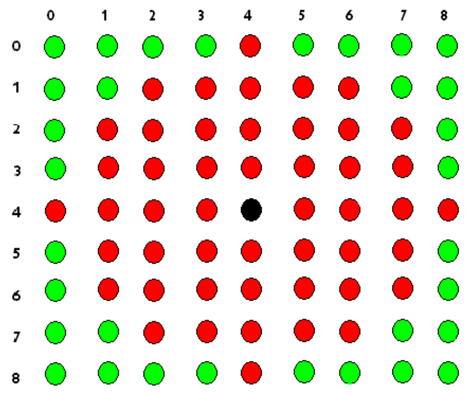
3) Experimental Study

In this section we will elaborately describe the graphical interface we use in our system and the process of getting approximately a circular movement.

3.1 Representing Circles in the Domain

Our area is represented by a matrix (2 dimensional arrays) in which we save the pheromone level and other vital information, where a forbidden area is marked with a certain value.
As it is known, the area covered by the robot is circular; hence we need a useful method in order to decide if a certain point is included in the circle the robot is covering, the proposed algorithm is to check the Euclidean distance between two points; the current location of the robot and the point to be covered, if the distance is bigger than the robot's radius then the point is not included in the circle.

For example:
Given a 9x9 matrix, a robot at point (4,4) with covering radius 2, the **RED** points are included in the circle, the **GREEN** are not.



3.2 The "Stairs Problem"

The representation of the circle in the domain looks more like stairs in the borders of our circle representation, graphically it is far from being a circle, therefore we should find a method that properly solve it.

3.2.1 Proposed solution and finding the appropriate radius

To overcome this problem and reduce the error in the calculation of the area covered by the robot, we must increase the number of pixels represent the domain, each pixel is to deploy into smaller squares. This will result in smaller "stairs" on the borders of the circle giving us a good approximation of circle shape.

Let **C** denote the size of the area.

Let **R** denote the radius.

The solution was implemented in the following way; we loop over range of radiuses.

Every step we increase R, we also deploy each point into a smaller R x R square, therefore the new size of the environment the robot moves on in each step is $C * R^2$.

This means that we also keep the ratio of the area radius at any step.

Assuming that the cycles are "pure cycles", since the area size grows in a constant ratio, then if we divide our area into equal-size circles that their size equals the size the robot's cyclic step covers we get an equal number of such circles in each step, in other words, every iteration the number of steps should be equal, hence we seek the radius from which the number of steps in the upcoming iterations start to be constant.

This equality of the number of steps we get when the shape that represents a circuit in our simulator is approximately a circle.

As we increase the area size and the radius in the same ratio we can stop iterating and choose the radius from which the number of steps starts to be constant.

<u>**Results graph:**</u>

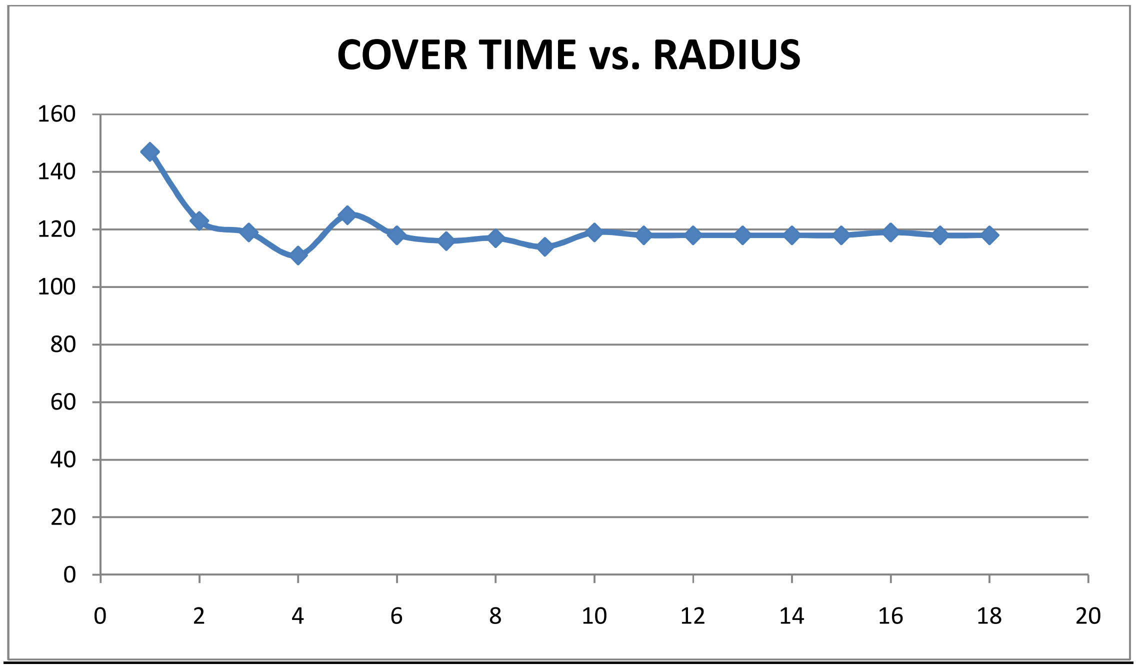


Figure 1: X– Radius variable; Y- Covering time average over 150 runs, at each step the area size is $C * X^2$.

We can easily notice that starting from radius 11 the covering time starts to be constant so from now on we use this radius in our upcoming analysis of different environments.

3.3 Cover Time vs. Area problem

In the previous section we investigated the cover time function behavior where domains have the same size but different shapes; in this section we will investigate the cover time function's behavior on areas that have the same shape but different sizes.

Naturally, when we increase the area size the cover time will be increased accordingly, but our investigation will be based on functionally how it will be increased.
Our experiment is based on investigation of squared areas where we linearly increase its ribs in each phase.
In each phase we run the robot on the current squared area saving its cover time to be analyzed later.

Phases of the experiment are as follows:
Let C denote the magnification factor of the area.
Let X(k) denote the part that represents the radius of the rib of square number k.

Initial phase:
X(1) = r (where r is the radius found in section 3.2)

Phase k > 1:
X(k) = X(k-1) + M (where M is a fairly small constant).

In phase number k the domain is a square of size: $\sqrt{C} * X(k)\ x\ \sqrt{C} * X(k)$.

Note: we repeat the operation above to the largest size we can run the simulator on (due to run time limitations).

3.3.1 Cover Time Anticipation

Obviously, as we increase the squares in each phase the cover time will increase respectively.
Note that the difference in sizes between phase k+1 and k subjects to the following expression:
$C * (r + M)^2 - C * r^2 = C * [r^2 + 2 * M * r + M^2 - r^2] = C * [2 * M * r + M^2]$
C and M are constants, the difference is linear!
Moreover, the area covered by the robot in each step is exponentially increased from phase to phase, so if we take a look at the additional part in the square comparatively to the previous phase we can realize that the additional circles number is in the order of the following expression:
$C * [2 * M * r + M^2]/(pi * (FOUND\ RADIUS)^2)$ where FOUND RADIUS is the radius found in section 3.3.1.
The expression above is linear, and then the Cover Time vs. Area function will likely be linear too.

3.3.2 Simulation Results

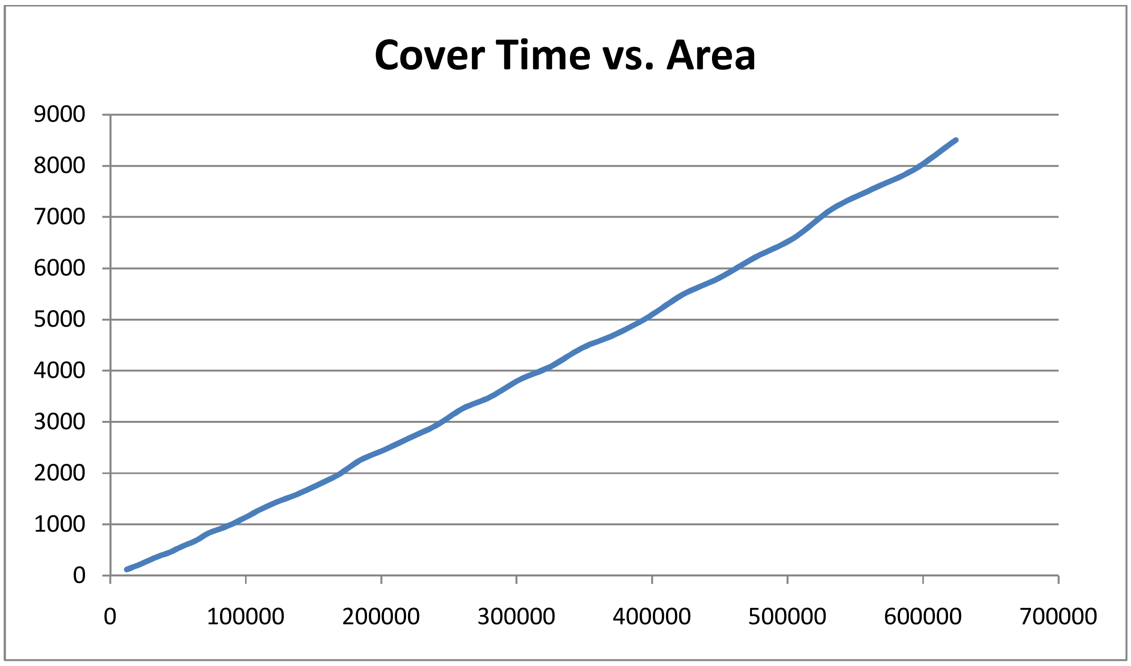


Figure 2: X- Area size, Y- Covering time average over 150 runs.

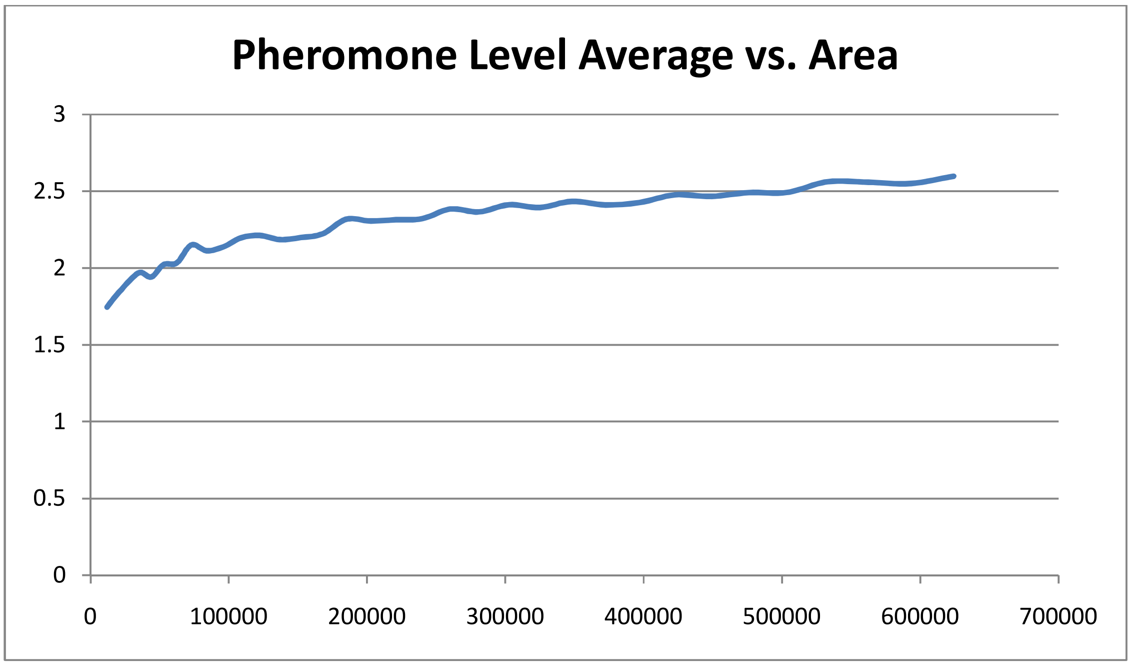


Figure 3: X- Area size, Y- Pheromone level average over 150 runs.

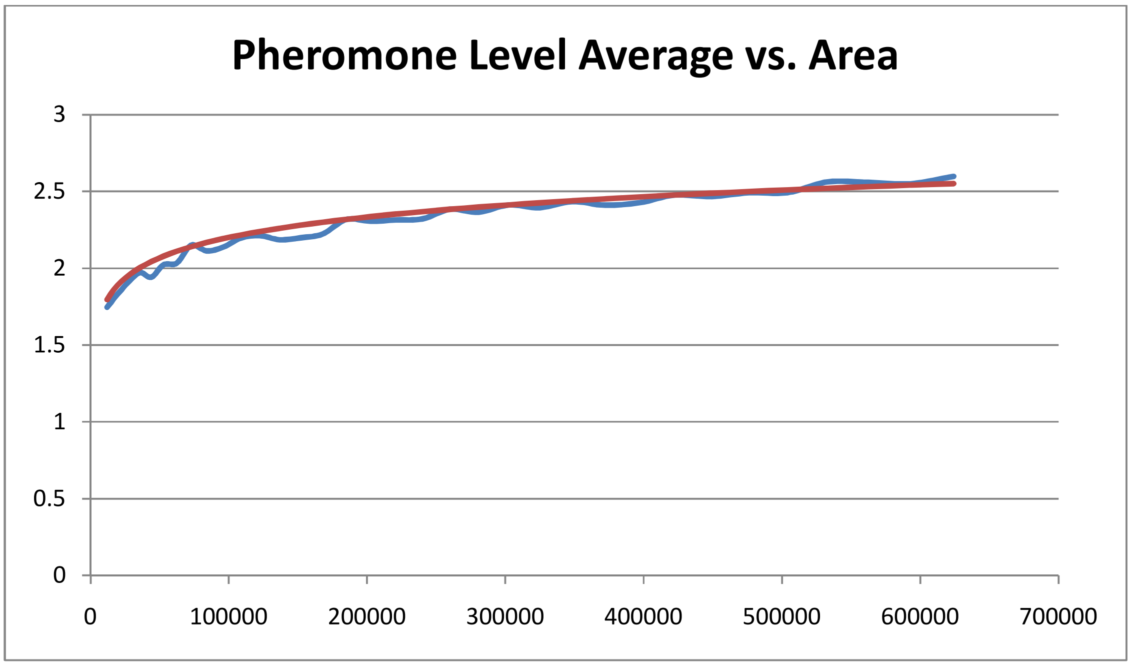


Figure 4: Blue function- Cover Time vs. Area function (Figure 2), Red function- y = 0.44*log(n). n- The number of points in the domain

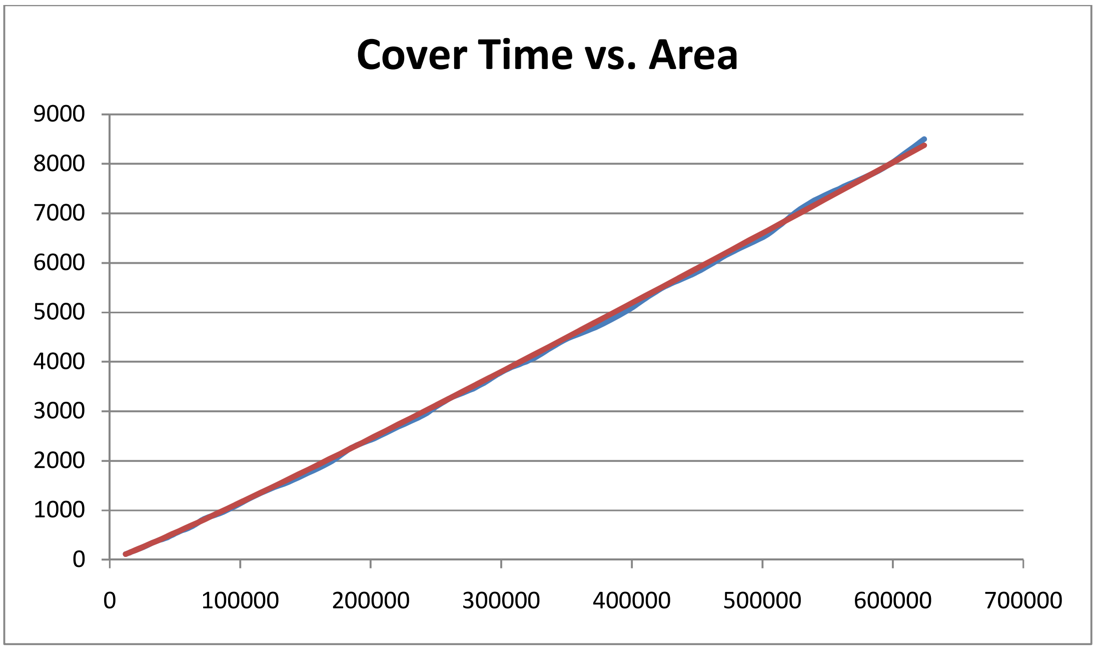


Figure 5: Blue function- Cover Time vs. Area function (Figure 2), Red function- y = 0.88*(A/a)*log(A/a). A- size of environment, a- area covered by the robot in a single step.

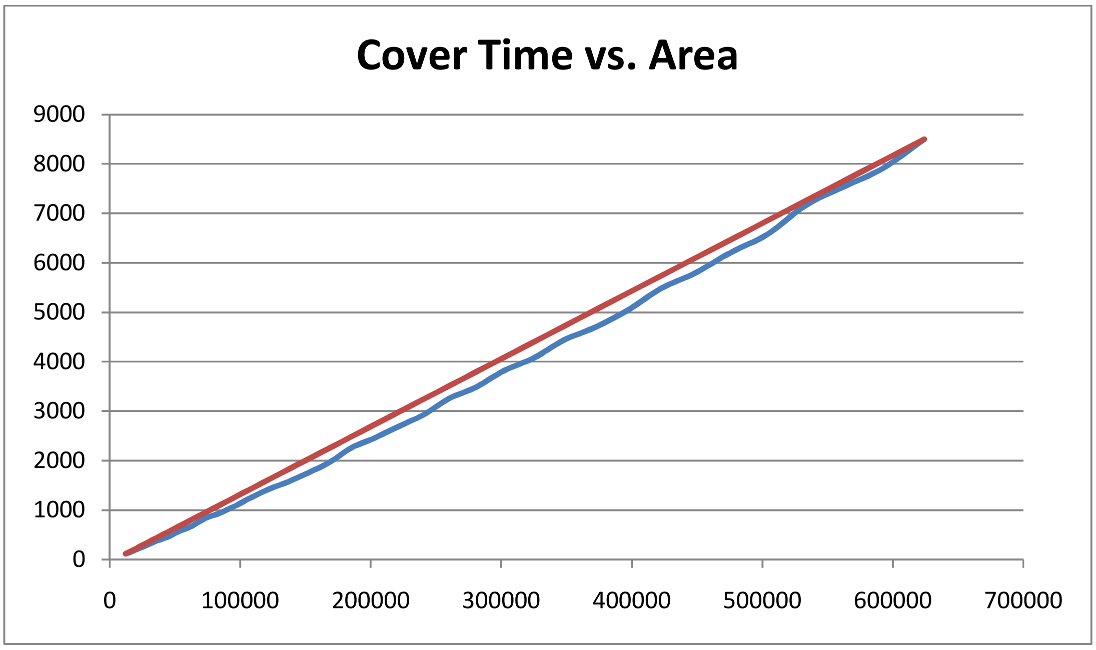


Figure 6: Blue function- Cover Time vs. Area function (Figure 2), Red function- y = 0.0137042484*x-49.8214264 (The linear function gradient was calculated using the first and last points)

3.3.3 Conclusions

- In section 3.4.1 we expected a linear behavior of the Cover Time vs. Area function, but as we can see the C*(A/a)*log (A/a) function gives as a better approximation of the robot's behavior (can clearly be noticed in Figures 5 and 6).
  When every point is visited approximately log(n) times and there are n points it is most likely that the cover time will be n*log(n).
- Looking at the Pheromone Level Average vs. Area function (Figure 3) we can see that as the square gets bigger the pheromone level is increased respectively; the probability of visiting the same point repeatedly is increased as well.

3.4 Cover Time vs. Environment Diameter

Theoretically, given the covering radius of the robot on the environment we are asked to cover, we can calculate the minimal number of steps (Cover Time) the robot needs in order to “clean” it.

Practically, from the robot’s point of view, it has no information about the area it is supposed to cover, thus we can assume that the cover time will exceed the minimal time, in addition thanks to the fact that the next step is chosen randomly in most cases.

Now the question that arises is **given an area, is it enough to determine the upper/lower bound of the cover time or the shape of the area has an effect also?**
In case the shape of the area has an effect, then what is the actual impact?
How much time would it take for a robot located in a rectangular area to cover it, comparatively with the same robot located in a squared area?

In order to answer the questions above we conducted the following experiment, we run our robot on rectangular areas with different width and lengths but in same sizes.

Phases of the experiment are as follows:
Let X(k) denote the length of the rectangle number k.
Let Y(k) denote the width of the rectangle number k.
Let A denote the area size (the area is fairly large).

We start our experiment with equal X, Y values, which means X(1) = Y(1) = $\sqrt{A}$.
In the upcoming steps where k > 1 the following is satisfied:
X(k) = X(k-1) – M (where M is a constant that satisfies: M<< $\sqrt{A}$).
Y(k) = A / X(k)
We repeat the operation above while X(k) > 1.

Note: in every step the area size is constant and equals: X(k) * Y(k) = X(k) * A / X(k) = A .

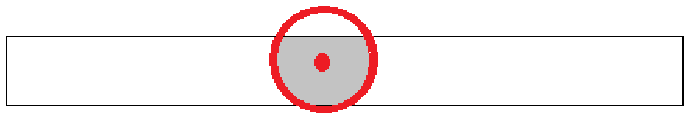
3.4.1 Cover Time Anticipation

The more we stretch the rectangle; logically we limit the movement of the robot to a smaller number of directions as the height gets smaller.

Thus we increase the probability of visiting the same points, hence the number of steps is supposed be increased respectively.

But it stops to get worse as the height of the rectangle starts to get smaller than the robot’s radius, because in this case the robot is bound to two directions at most, and in each step it covers the points in a way that it doesn't leave gaps vertically, and the result should be

2*(Length/Diameter) in average for a relatively small height, multiplied by two because in the begging the robot chooses one direction and when it reaches the rectangle's edge it will go back in the opposite direction to the starting point and from there to cover the second uncover part, graphically in each step we have approximately the following behavior:



Where the bigger red circle is the area covered by the robot in a single step, the red point is the current position of the robot and the grey part is the covered area in that step.

Important to denote that this is valid for relatively small heights!

3.4.2 Simulation Results

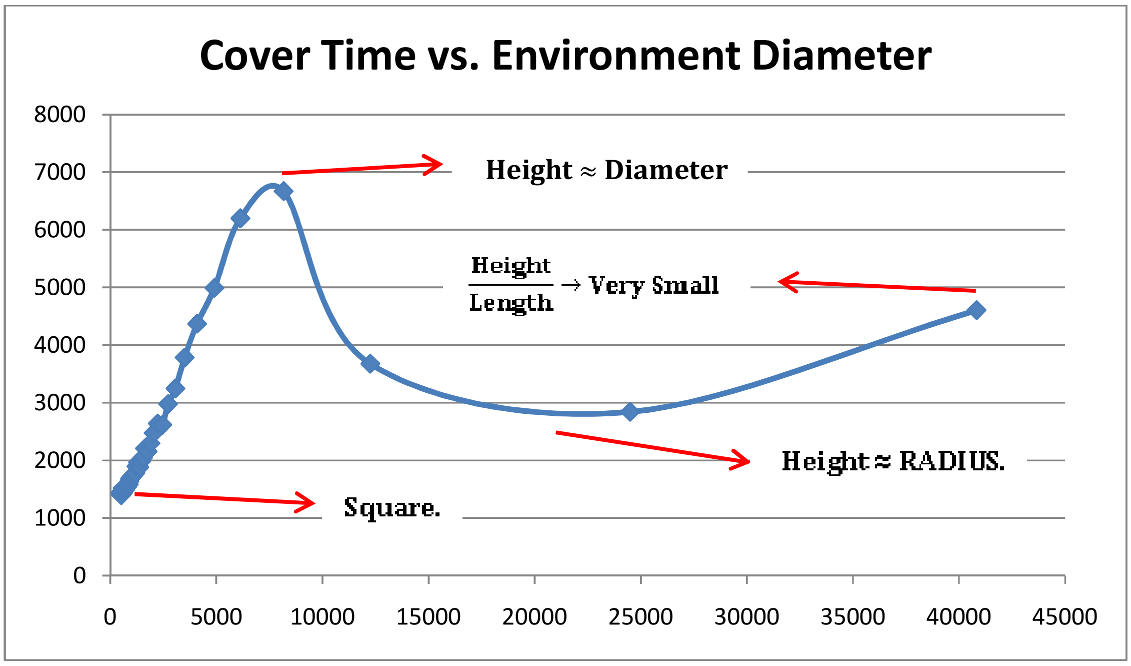


Figure 7: X- Rectangular environment's diameter (area size is 350^2), Y- Covering time average over 150 runs.

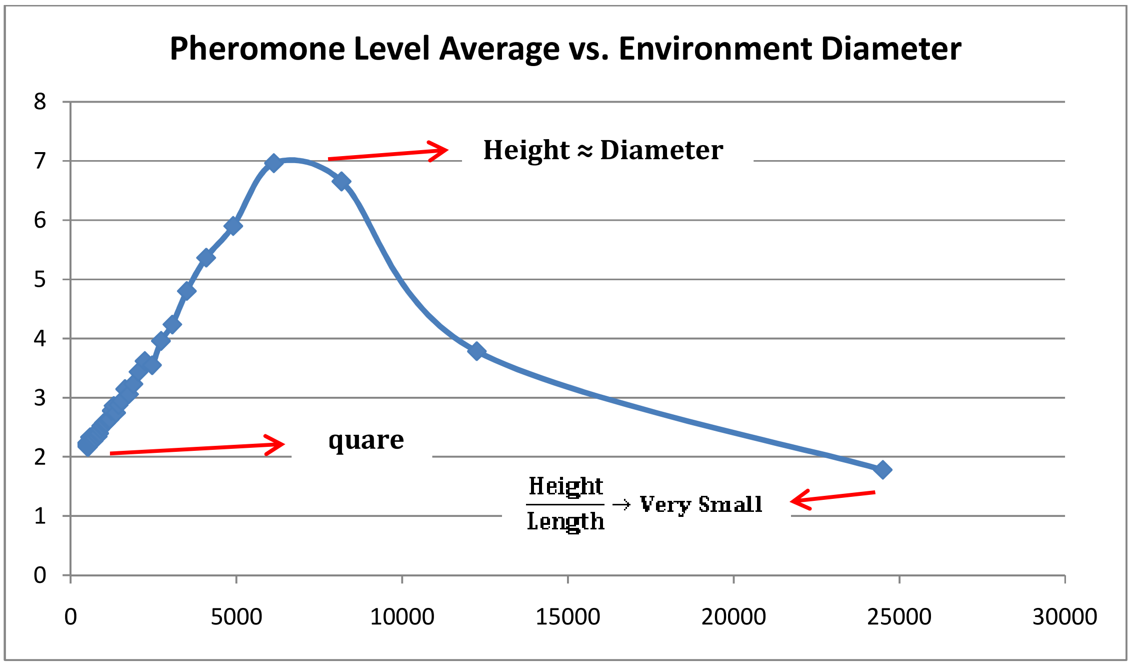


Figure 8: X- Rectangular environment's diameter (area size is 350^2), Y- Pheromone level average over 150 runs.

3.4.3 Conclusions

- The more the shape is squared, the less the cover time is.
  This is compatible with the following explanation: the more we stretch the rectangle the more we limit the movement of the robot, thus it will result in increasing the probability of visiting the same points repeatedly, and this is reflected in Figure 8 (the rise continues until the height is close to the diameter, this will be explained later in this section).
  In a squared environment the robot has more choices of directions which results in a low pheromone level average (see Figure 8).
  The continuous rise of the covering time sustains until the height is close to the robot's diameter.
- Following the rise of the covering time we get a drop in it as the height gets smaller than the diameter and a strict drop as it gets closer to the radius.
  As we get closer to the diameter and smaller, actually we increase the probability of getting two choices of directions of the robot's movement, by getting closer to the radius we also decrease the probability of getting gaps vertically, thus it will result in covering the area in a horizontal movement.
- The strict drop is followed by a strict rise in the covering time for very small heights.
  When the height reaches very small values the length gets very long respectively as the ratio between both begins to grow significantly.
  The robot's straight movement over a significantly long environment will result in increasing covering time as the ratio between the environment length and the diameter begins to be pretty large (notice that the increase in the covering time has no connection to the pheromone level, which strengthens our claim).
  Notice as the height reaches significantly small values than the radius, the covering time function will be committed to the following range:
  $\frac{Rectangular\ Environment's\ Length}{Diameter}$ < C.T < $2\ x\ \frac{Rectangular\ Environment's\ Length}{Diameter}$
  Where $\frac{Rectangular\ Environment's\ Length}{Diameter}$ is the best case we get when the starting point's distance from the edge is less or equal than the value of the radius, and
  $2\ x\ \frac{Rectangula\ Environment's\ Length}{Diameter}$ is the worst case that we get when the starting point's distance is (radius+1) and the robot's first choice of direction is opposite to the closer edge so the robot will have to cover each pixel until the far edge and back away to the second edge covering the remaining very few pixels (~radius).

3.5 Cover Time vs. Multi-Agents problem

In the previous sections we have experimented single agent behavior over various domains; in this section we will investigate the covering time average of a fairly large environment when it is being covered by multi-agents and not just a single one, introducing the covering time differences for various numbers of agents and the reasons for them.

Note: a single point cannot be occupied by two or more agents simultaneously.

3.5.1 Simulation Results

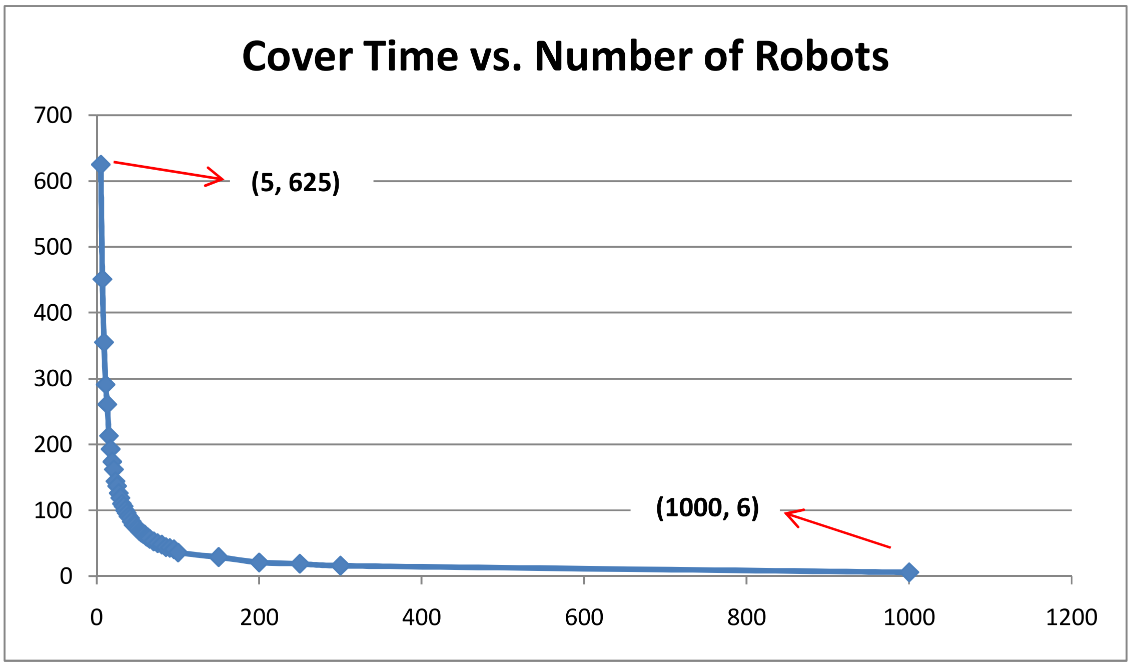


Figure 9: X- Number of Robots, Y- Covering time average (of squared environment of size 500x500) over 150 runs.

3.5.2 Conclusions

- Trivially more agents being used to “clean” the domain, the faster it will be covered.
- The Cover Time vs. Multi-Agents function starts to drop at a fast rate in the beginning but the drop stops to be so firm when there are a relatively large number of robots until it becomes approximately constant, the simple reason is that when we have more robots the ratio of the total area covered by them in one step and the environment is relatively high, as a result the chances of collisions are significantly increased.
  For example, when the environment is fairly large and we have two agents, the number of collisions is too low as they the probability to start from distant points is fairly large, thus we can expect that the covering time will be half of that it takes for a single robot to cover the area, but the probability of having distant starting points is decreased when we add more and more robots, causing collisions and as a result inefficient utilization of robots

3.6 Improving the Algorithm

After we had different kinds of tests in order to explore the robot's behavior and how the various parameters affect the covering time of our agent, our target in this sections it to introduce an improvement to the algorithm.

In the classic algorithm the next step was arbitrarily selected from a group of points with minimal pheromone value.

The algorithm is not optimal in some cases, for example when the agent is located in a domain where its scanning range includes holes (a very small number of adjacent uncovered points) and in a different area of the same range there are plenty of uncovered points, the inefficiency begins when the next step chosen by the robot is in the latter area of the scanning range, as a result we get a covered area with a hole in it.

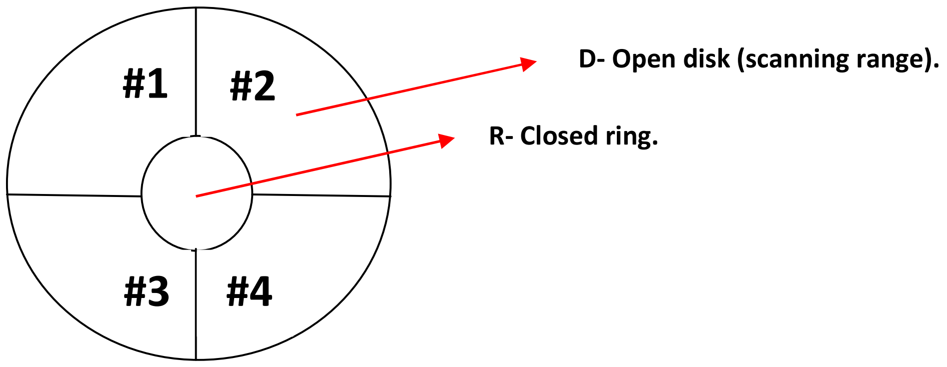
For this reason, the robot will have to find the left uncovered points in an arbitrary movement, thus causing increasing in the covering time and the pheromone level.

3.6.1 The Next Step Proposed Improvement

Because of the case mentioned in the previous section, hereby we propose a new technique of choosing the next step which should decrease the number of holes and as a result the covering time and the pheromone level as well.

The proposed change is in choosing the next step*:

We divide the open disk to four equal **regions** as follows:



We choose the next step according to the following algorithm:

- **Find r := a region (#1,#2,#3,#4 or NULL) with minimal positive numbers of uncovered points (pheromone level = 0).**
  /* NULL is returned in case there are no zero level pheromone values at any point. */
- **If** r ≠ NULL **then Find x := a point in R(r, 2r, p) && located in the found region with minimal value of σ(x, t).**
- **Else, Find x := a point in R(r, 2r, p) with minimal value of σ(x, t).**

*** Step (1) in the classic algorithm.**

According to the proposed change, the robot will first cover the areas with small number of uncovered points and just then it will move to clean the rest.

The holes are recognized by a small number of adjacent uncovered points, thus the proposed change will leave the environment with smaller number of holes in it.

Proving the correctness of the proposed algorithm:
Note that the algorithm never gets stuck and there is always a point that can be chosen as the next step: In the first three steps of the algorithm we always try to visit uncovered points to be covered in the next covering step of the robot.
If the algorithm doesn't find the next step according the first three phases of the algorithm above, the next step will be chosen according to the classic algorithm, thus the correctness of the new algorithm results from the proof of correctness of the classic algorithm.

Note: the algorithm never stops.

3.6.2 Simulation Results

In order to show the priority of this algorithm over the classic one, we introduce graphs including same parameters used in the experiments we first carried out, representing the new behavior of robot and comparing it to the old one.

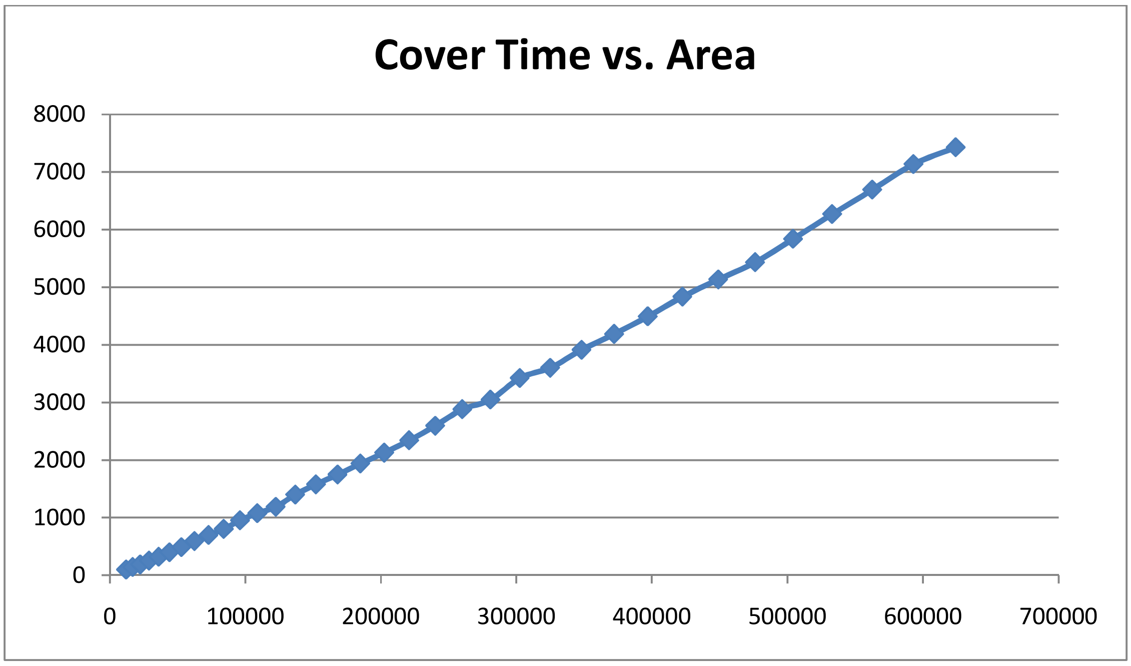


Figure 10: X- Area size, Y- Covering time average over 150 runs.

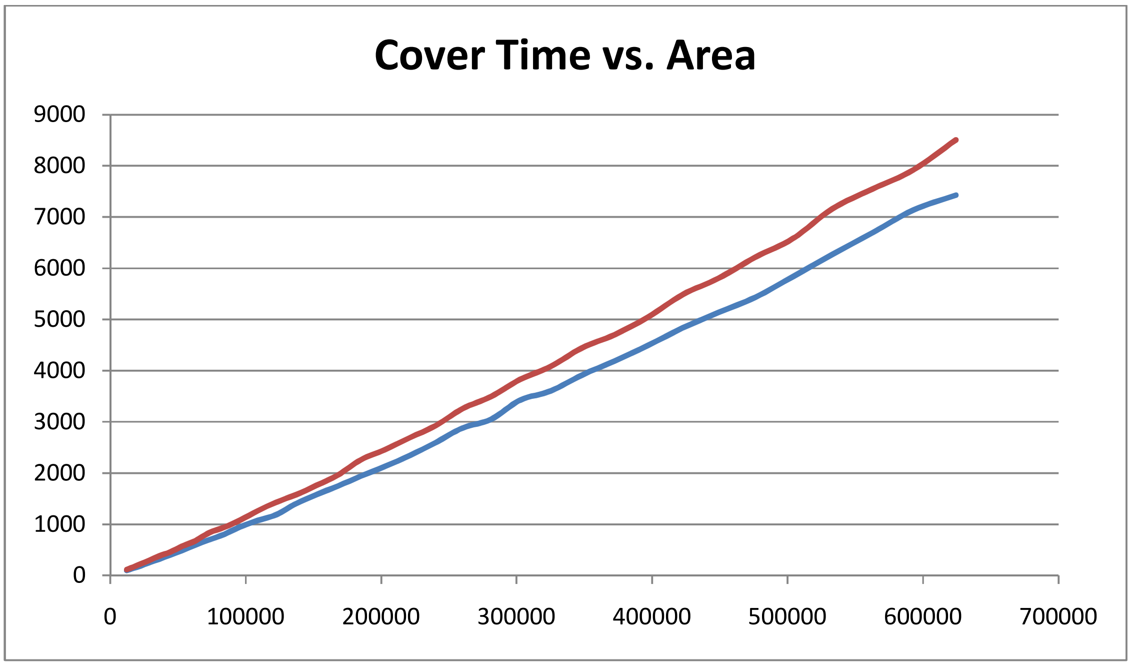


Figure 11: Cover Time vs. Area, Red function- Old algorithm's output, Blue function- New algorithm's output.

- Note that when the area size is big the difference between the two algorithm's gets bigger respectively, this is pretty logical because the more holes there are in a big environment the more it takes for the robot to find them.
  Unlike the old algorithm, the new algorithm avoids leaving holes and the mentioned above is reflected in their Cover Time vs. Area functions.

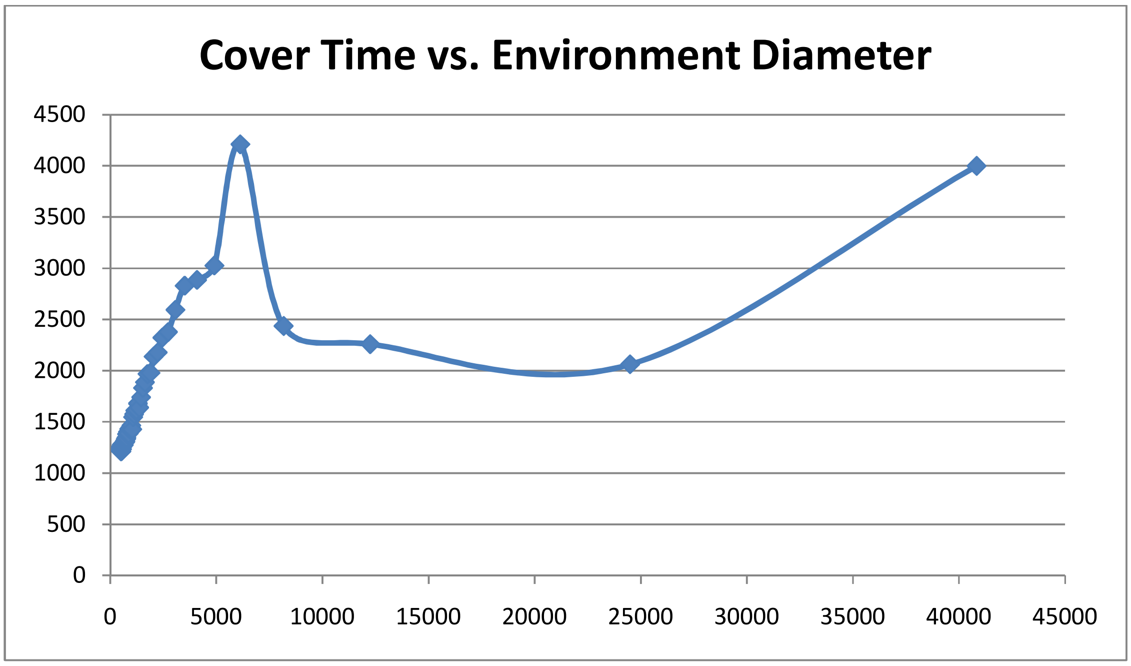


Figure 12: X- Rectangular environment's diameter (area size is 350^2), Y- Covering time average over 150 runs.

- Note that the function behavior is similar to the old algorithm's behavior, which strengthens the claims mentioned in section 3.3.

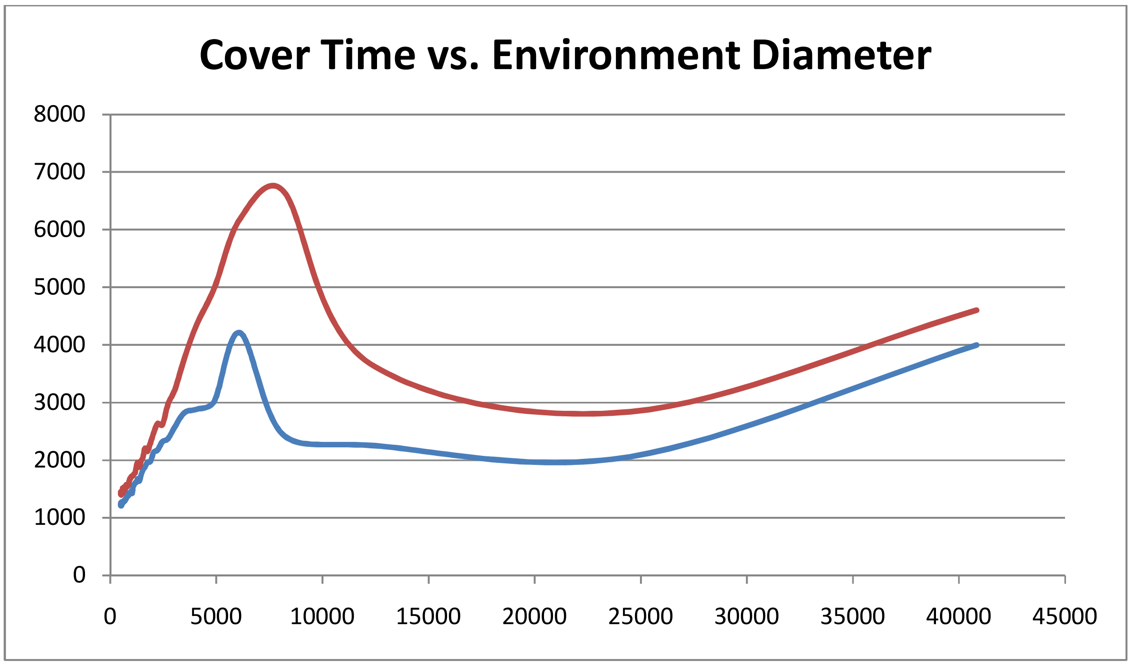


Figure 13: Cover Time vs. Environment Diameter, Red- Old algorithm's output, Blue- New algorithm's output.

4) Conclusions

Here we will summarize our test results introducing the general conclusions about the agent(s) behavior on the various environments we explored.

- We can deduce from the Cover Time vs. Environment Diameter results graph that the more the domain is squared the better the cover time is.

- We can calculate the Cover Time of a single agent running on a squared area with a constant size increasing factors (e.g. adding a constant to the square's rib each time) by the C*(A/a)*log(A/a) formula that can be calculated using the results of few runs of the algorithm over the claimed areas.
  It will obviate us from wasting time running the simulation when it comes to relatively large domains and the results are approximate as we can see in the former sections.

- Looking at the Cover Time vs. Number of Robots we can deduce that adding more and more robots will definitely begin to be inefficient at some point where the number of collisions starts to grow up, which means at that point we are adding robots in vain, poorly exploiting resources and misusing functionality.

- After introducing the new technique of choosing the next point and graphically comparing it to the old technique, we can obviously see the significant drop in the covering time independent of the shape and size of the domain, thus it is worthwhile to use it as a substitute to the old one.