Bachelor Thesis

7.5 ECTS

Ant coverage using diffusion at and between grids

Martin Borgt
3939707
Utrecht University
m.j.a.a.borgt@students.uu.nl

Supervisor:
Gerard Vreeswijk

April 12, 2016

Contents

Abstract

This research looks into how an ant robot using Node Counting without diffusion compares to one using Node Counting with diffusion at repeatedly covering a generic grid and the bottleneck sub-grid. This is done by running a simulation in which the agent, an ant robot, covers the bottleneck sub-grid an increasing number of times and sets 100000 steps for 100 runs in the maze grid. The bottleneck sub-grid is a three armed square that connects three otherwise unconnected sub-grids together. The maze grid is an open grid where 20% is replaced by walls which the agent has to maneuver around. Efficient coverage on the bottleneck is related to efficient coverage of grids that contain a similar crossroads. The average time between visits is similar for both algorithms, but the average standard deviation is lower for Node Counting with diffusion, which means that the grid is visited more regularly. When first covering a bottleneck the Node Counting algorithm without diffusion achieves a success rate of 0.83, versus 0.75 for the same algorithm with diffusion. When repeatedly covering the bottleneck sub-grid, the success rate of Node Counting without diffusion converges to 0.71 with diffusion it converges to 0.34, so diffusion has a negative effect in this case. So, whether diffusion is beneficial is dependent on the shape of the grid.

1. Introduction

Continuously patrolling an area is a task we see robots performing more and more. Be it an autonomous lawnmower, a vacuum cleaner, an environment monitor or a security patroller. These robots do not need to find their way perfectly, as long as they get to every part of the area in acceptable time and continue to do so. They all have in common that they need to perform the same task at every destination and that getting there has to happen in a non-intensive and quick manner.

When designing a simple and efficient robot, it is often impractical to try to map the environment and try to build a plan on incomplete information. This is computationally intensive and a being misplaced for any reason makes the robot start again. Ant robots are able to overcome these difficulties by following a simple set of rules and marking the environment. Because these robots do not need to know where they are and how far away areas look, they do not need to be very precise in their work.

Ants covering an area in nature use pheromones to make add these marks to their environment. This way multiple ants can work together and easily indicate in which direction they no longer need to search, by avoiding places containing pheromones.[1] These pheromones are not placed to stay in an exact area, but rather disperse in a way their concentration is highest at the center. The conventional way of implementing this only leaves pheromone at the position the ant robot is at. Not much research has been done into algorithms that use diffusion or dissipation of heuristic values to adjacent areas to have information spread further than the exact location of the agent, while it is an interesting mechanism that is already used in nature. Implementing diffusion of the pheromone may provoke significantly different behavior in the ant robot.

A very often used algorithm when covering a space is Node Counting. This consists of simply tallying the number of times a place is reached and always choosing to go to the place that holds the lowest tally or, amount of pheromone. To the spreading of this pheromone to nearby areas will be referred as diffusion. This research investigates if diffusion makes an impact and if it does, how big that impact is, on an agent's ability to frequently cover an introduced sub-grid.

Relevance for Artificial Intelligence

According to the Brittanica encyclopedia, Artificial intelligence means: the ability of a computer-controlled robot to perform tasks commonly associated with intelligent beings. Even though it was just stated that efficient coverage is a task easily performed by insects such as ants, it is a challenge that falls within the scope of artificial intelligence. One reason for this is that the robots communicate. They do not do this directly, but indirectly via the pheromone left on the grid. They communicate to their future selves and to other robots to remind them of where they have already been. They also make decisions of where to go based on these communications, even if they have never been in that position before. This adaptability to circumstance combined with the ease of adapting to errors in communication, pheromone levels or a change in the environment intertwine this area of research truly with artificial intelligence.

Research question

This research compares the behavior of a simple Node Counting algorithm that does not include diffusion of the marked values it leaves on the grid to the behavior of one that does. These values signal the amount of visits that have been made to a particular square on this grid. How the performances of these algorithms are compared is explained in the Methods section.

The research question is the following: "What are the effects of diffusion of pheromone in node counting on recurrence on a grid?". To answer this two sub-problems are considered. How these algorithms compare on a random grid that has many possible paths from each point to every other will be investigated and how they compare when grid section are separated from each other in every way but a small pathway will be looked into by making use of a sub-grid that does just that.

The way the effects will be presented is in the form of average time between visits on the random grid and in the form of success rates on the bottleneck section. These success rates correspond to the percentage of times the agent visits both other sub-grids connected to the bottleneck before it returns to the sub-grid in entered the bottleneck from. The bottleneck sub-grid and other concepts used here will be defined in the Concepts section.

The research question will be properly answered when we can tell if one of the algorithms performs better when repeatedly covering a grid that looks like the randomly generated grids and if one performs better when covering the separated grids. Since it is impossible look at infinite coverage, the grid will be cover once at the start and then the number of coverages is increased to see if we can estimate how efficient both algorithms behave at infinite coverage.

2. Concepts

This chapter introduces some of the concepts that are used in this paper. First, the type of grid used is explained along with a more precise definition of the maze the first part of the experiment is conducted in. The agent is introduced next, followed by the specific algorithms it makes use of. Last, the bottleneck sub-grid is defined.

The grid

The four-connected grid consists of a number of squares that are all connected to at least one and up to four adjacent squares. An open grid is identical to a von Neumann neighborhood[6] and the way the used grid differs from an open grid is that on any square, movement in up to three of the four possible direction may not be possible. This way, the grid can represent a maze, a map or any 2D representation of space. The agent can reach each square from each other square by traversing from one square to a horizontally or vertically adjacent square and continuing this process from the initial square to the final square. The cost for moving from one square to another is the same for each square. Each square can contain any positive number value placed there by the agent. One grid that is used in the simulation is randomly filled with barriers in order to form a maze, the other contains at least one bottleneck, these entities will be explained later in this chapter.

The maze

The maze is a version of the earlier introduced grid. The maze used in this simulation is a square 20 x 20 grid consisting of 400 squares. 80 of these squares are then turned into walls by the mazebuilding algorithm, the wall squares are indicated by a different color than the grid. The mazes are constructed by randomly adding an agent to the grid facing a random direction. Then this agent is allowed to turn the square they are on into a wall and take a step forward if there are no adjacent squares that are already turned into a wall or teleport to a random square on the grid that has no adjacent walls. This step is repeated until enough walls have been placed. This guarantees that the entire grid remains traversable around both ends of the wall. The walls form obstacles the agent has to find its way around efficiently, but not force the agent to take a single path.

The agent

The algorithms will be applied by an agent whose goal it is to explore the entire grid as efficiently as possible. A grid is fully explored when the agent has visited every square in it at least once. The agent is only able to look at the square he is in and the squares adjacent to it. He can add a marking to a square increasing its marked value any and see marked value of the square it is in. These marked values can diffuse to adjacent squares, adding a value lower than the value in the square itself to the surrounding ones. The agent uses the value on each square to decide which square he will go to next.

The Node Counting algorithm without diffusion

The first algorithm introduced is called Node Counting. Node Counting without diffusion is shown first in pseudo code and then this pseudo code will be clarified in order to explain some properties of this algorithm.

Initially, all squares are marked zero.

1. The starting square of the agent is s.
2. The agent looks at all adjacent squares and determines which one has the lowest marked value, ties are broken randomly. This square is s'.
3. The agent adds one to the value on s.
4. The agent moves to s'.
5. s' will from now on be referred to as s.
6. If all squares have been visited at least once, stop. Otherwise replace s with s' and go to step 2.

When the agent makes use of this algorithm to decide which square to visit, the agent will visit the adjacent square with the lowest marked value. It will then add one to the marked value of the square it leaves.[5] Changing this value of one to any amount does not change the algorithm, since value added can always simply be brought back to one by dividing the total marked value on a square value added per visit. The value a square holds is the same as the number of visits that have been made to a square when each added value is one. All unvisited squares will be marked zero when only the Node Counting algorithm is used. This makes the Node Counting algorithm always choose an unvisited square over an already visited square when the option is available.

The Node Counting algorithm with diffusion

Adding diffusion to the Node Counting algorithm only changes step 3, where the marked value is added to the square. Instead of only adding the value of one to the square the agent will leave, all squares adjacent to the square the agent will leave will have a value $0 < \text{value} < 1$ added to them, 0.1 in this case. Since all relevance of marked values is relative to each other, it is not a problem that the total value added at each step is larger in this algorithm than in the version of it without diffusion. Below is the pseudo code for this algorithm.

```
Initially, all squares are marked zero.

    1. The starting square of the agent is s.
    2. The agent looks at all adjacent squares and determines which one has the lowest marked value,
       ties are broken randomly. This square is s'.
    3. The agent adds one to the marked value of s and 1/10 to all squares adjacent to s.
    4. The agent moves to s'.
    5. s' will from now on be referred to as s.
    6. If all squares have been visited at least once, stop. Otherwise replace s with s' and go to step 2.
```

There is a slightly easier way to implement the visit count algorithm with diffusion. This version does not diffuse the marked values when placing them in the environment, but instead deduces the amount of diffusion from surrounding squares. This version of the algorithm can be used when simulating diffusion, but not when using the agent in an actual environment, because the agent would need information from more squares than just the adjacent squares to choose its next step. In this alternative version, only step 2 is changed from the visit count algorithm without diffusion. The algorithm sums the marked value of the diffusion squares, multiplied by the marked value on adjacent squares value on a square, before comparing it to others in this step. Since the marked value on a square is equal to the amount of visits that have been made to the square, the diffusion from that square into any adjacent square is equal to the marked value on it multiplied by the factor of diffusion. The pseudo code for this algorithm follows below.

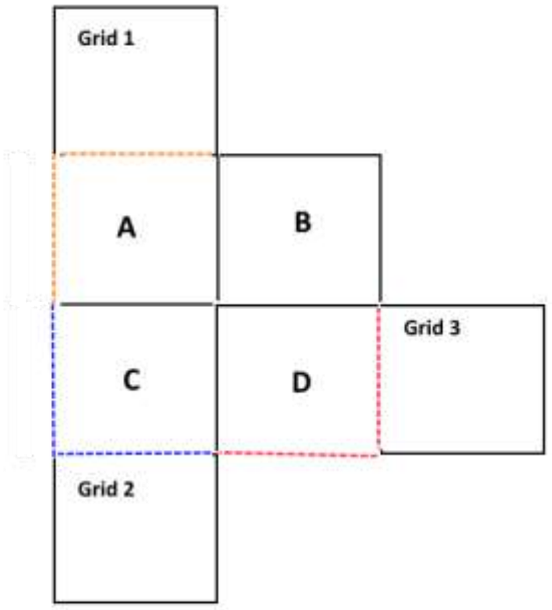
```
Initially, all squares are marked zero.

    1. The starting square of the agent is s.
    2. The agent looks at all adjacent squares. For each square, it takes the marked value m and the
       marked value of adjacent squares n. It then adds n divided by ten and m together and
       determines for which square this results in the lowest value, ties are broken randomly. This
       square is s'.
```

3. The agent adds one to the value on s.
4. The agent moves to s’.
5. s’ will from now on be referred to as s.
6. If all squares have been visited at least once, stop. Otherwise replace s with s’ and go to step 2.

Definition of the Bottleneck

The bottleneck is a special case of a connection between parts of a grid. It is a connection between three grids that are not otherwise connected to each other. The reason this number of grids are selected is that it is the minimum amount needed for interesting behavior. When a one square wide connection between two grids is investigated the path the agent takes when the grid is unvisited will always go from one to the other. When the agent has to pick between two possible paths, it always chooses at least one path, but sometimes returns the way it came without visiting the other path. The same thing happens when more than three sub-grids are considered, but this is merely a more complex version of the same problem. The first connection is from a sub-grid to the bottleneck. This connection has a width of only one cell, which means that whenever the agent traverses this connection it has to cross this one cell. This attaches to a square of four cells, which branches of into two sub-grids, again with a connection with a width of one cell. These two branches connect to two adjacent cells on the square that are not the square the main grid is connected to. An example is found in Figure 1. In this, the grids branching off are Grid 2 and Grid 3.

Figure one: The Bottleneck



The cells named Grid (number) are single cells connected to a sub-grid that is not otherwise connected to the other cells. The cells A, B, C and D together form the square that connects the grids. The grids are connected to the squares by a dashed line. Different versions of the bottleneck that are equivalent in this paper can be constructed by instead connecting the grid to the other dashed line of the same colour, rotating the figure, or mirroring it in any direction.

3. Methods

In the methods chapter, the way both algorithms are compared are stated and the reasons for choosing this particular type of grid are explain. In order to explain this, criteria are formed that decide whether an agent is successful or unsuccessful at efficiently covering the grid. The way the simulation is set up is explained after this using Figure 1 and referencing the simulation software used.

Comparing performance in a random maze

There are two simple ways in which the performance of the agent using each algorithm to cover the maze can be measured. The first way is counting the average number of steps it takes an agent to cover the grid a certain number of times. This way the average number of steps an agent has to take to cover the grid can easily be calculated by dividing by the number of times the grid has been covered. This measurement method is not used because it gives no information about how the grid is covered. When a grid is covered by going up and down between two squares the required number of times and then iterating this step over two new squares, the average amount of visits per coverage may be very low, but it is clearly not the repeated coverage that is sought. This is an extreme case, but the fact that information on how the grid is covered is necessary to be able to judge the algorithms.

The measurement method that is used, is the average time between visits. Whenever the agent visits a square, the amount of steps that the agent has taken since it last visited the square is used to compute the average. The standard deviation in this average will tell us if the squares are visited regularly.[3] A high standard deviation means that the amounts of steps between visits vary a lot, while a small standard deviation indicates that the agent visits the squares with similar amounts of steps between each.

Why make use of the bottleneck?

It is unknown whether an algorithm using diffusion performs better than a similar algorithm not using diffusion, but it seems likely when looking at examples in nature. Insects often make use of pheromones to communicate the location certain entities such as food. It is also difficult to avoid diffusion of heuristic values when they are left on the terrain that is being covered. Knowing whether diffusion would help in this specific case would aid in choosing a good coverage algorithm for covering similar grids in the future. One reason for choosing the bottleneck is that it looks like a connection where the difference in performance of Node Counting with diffusion and Node Counting without diffusion is small.

A reason for choosing the bottleneck sub-grid is that good coverage on this sub-grid implies good coverage in general. Since Grid 1, Grid 2 and Grid 3 are not connected via any other paths besides the bottleneck, an agent arriving at the bottleneck from Grid 1 needs to cover Grid 2 and Grid 3 before returning to Grid 1. The agent will travel past the square connecting Grid 1 to the bottleneck and back to it in what we define as one visit. A visit is started when the agent travels from Grid 1 to square A and ended when the agent travels from square A back to Grid 1. Once when entering the bottleneck and once when leaving it. Because of this, the agent only gets another chance to visit Grid 2 and Grid 3 when it again enters the bottleneck, which is not dependent on whether it has visited the other sub-grids on its last visit. For this reason, we will refer to a visit where the agent reaches both Grid 2 and Grid 3 as a successful visit and we will refer to a visit where at least one of these is not visited as an unsuccessful visit.

Simulation

Our simulation is run using the Netlogo simulation environment. The reason for using this program is that is easy to use when simulating a process over time. The space that is discovered by the agent is easily manipulated and can be adjusting during the simulation using sliders and choosers. Its BehaviorSpace automates iteration over different inputs and lets the user create a table from the values that result from the simulation. Netlogo is often used when simulating robots covering space because a functional grid that can be covered by an agent is readily present beforehand, making programming the simulation come down to applying the algorithms used in it. The script that was used to run our simulation can be found in the appendix.

Maze

This simulation is first executed using the Node Counting algorithm without diffusion and then it is executed using the Node Counting algorithm with diffusion.

The maze is generated in the way described in the Concepts section. Once the desired amount of wall squares is reached the generation of walls is stopped. The agent is then placed on a random square that is not a wall. The agent will continuously cover the maze until it has made 10,000 steps. Using the BehaviorSpace, this is performed 100 times per algorithm to ensure that the average is taken over different grids. The output will be the average time it takes the agent to return to a square and the standard deviation in that time.

Bottleneck

The bottleneck grid is represented in this simulation in the same way it is shown in Figure 1. In this representation Grid 1, Grid 2 and Grid 3 consist of three squares in a row. This to ensure that the markings on the squares adjacent to the bottleneck represent the correct number of times an agent has travelled to it. The agent will start a simulation run in the square in Grid 1 that has only one adjacent square.

The agent will start by making a run that lasts one visit using first the Node Counting algorithm without diffusion and later using the Node Counting algorithm with diffusion, the amount of diffusion added to each adjacent square will be 0.1 times the value the agent adds to the current value of the square it leaves.

Grid 1, Grid 2 and Grid 3 represent random sub-grids, with the assumption that the agent starts in Grid 1 and that Grid 1 is larger than Grid 2 and Grid 3. In the simulation, the agent does not actually cover Grid 1, Grid 2 and Grid 3. It only arrives at a square that represents the agent covering these grids. For this reason, a small change is made to the algorithms to ensure that they behave correctly. This paper does not look into how the agent performs at different types of sub-grids and that is why we have decided to generalize the simulation to work with any sub-grid. We will increase the visit count at the ends of the three square sub-grids with 2 instead of 1 when the agents exits the square. This is the values we would expect in this square when the agent has travelled over it once when it from the end of the sub-grid enters bottleneck and once when it exits it again. By doing this, the achieved result is the same as it would be if the agent travelled to a sub-grid that continued from the end of the sub-grid.

A visit is ended and a new one started when the agent arrives at Grid 1 from square A. After this run it is decided whether it was a successful visit or an unsuccessful one. This run is repeated for each algorithm until the percentage of successful runs converges. Using the BehaviorSpace, this process will be repeated from one run with an increasing number of runs for both algorithms until the average number of successful visits per total number of visits converges at 200 runs. The output generated is the algorithm and the number of runs connected to the achieved success rate.

Assumptions

In the simulation, a number of assumptions are made. These assumptions are listed below.

- Our first assumption is that the agent always starts in Grid 1 when covering the bottleneck. If the agents starts in Grid 3, the sub-grids can be renamed to let the agent start in Grid 1 instead. Cases where the agent starts in Grid 2 or in the bottleneck are not taken into account. When the agent starts in Grid 2, the agent will always make a successful first run, this does not provide much information on which of the algorithms works better.
- Another assumption is that at the start of a simulation run, all squares are unmarked and when the agent marks a square, these value added will always be of the exact intended size. This to make sure that the agent applies the algorithms the way they are intended. If the agent were to consistently add wrong values this might affect the validity of the comparison between the algorithms.
- The agent is always perfectly able to detect the sum of the marked values on all adjacent squares and always perfectly marks the square it leaves. If intended, it marks adjacent squares too. This is assumed to make sure the agent performs the algorithms the way they are intended.
- The agent can never unintentionally travel to another square and will always successfully travel to an adjacent square if it is their intention to do so. In practice, this may not be the case. Investigating the effect of errors on the Node Counting algorithm without diffusion is done in different researches.
- The agent is able to completely cover a sub-grid in the bottleneck in one go if he enters it. If the agent is not able to always cover these sub-grids in one go, he would have to bring multiple visits in order to make a successful run. The agent may not actually be able to do this, if it fails at covering a sub-grid before returning, the run might not be successful.
- In the bottleneck, Grid 2 and Grid 3 will take a lot less time to cover than Grid 1. A successful visit is one where both Grid 2 and 3 are reached before the agent returns to Grid 1. Therefor the agent might visit Grid 2 or Grid 3 multiple times before returning and still make a successful visit. If these visits take a longer time than exploring Grid 1, the agent is not exploring the grid efficiently while making successful runs.

Representation of results

The values resulting from the maze simulation are simply shown in a table, Table 1. The values resulting from the bottleneck simulation are represented using R. The table that results from the Netlogo experiment is loaded into R, the first six lines contain information we have no use for, so these are skipped. An image is created where all resulting plots will be displayed in. Dashes and quotation marks are removed from the dataset in order to use values more easily. The data is then separated into a set containing the Node Counting algorithm without diffusion values and one containing the Node Counting algorithm with diffusion values. The percentage of successful visits are plotted on the y-axis which is scaled 0 to 1 against the total number of visits on the x-axis for one of the selections made. Then, using the "points" command the other dataset is plotted in this same window. This results in Figure 2. The .R script that is used for this visualization can be found in the appendix.

4. Hypothesis

The hypothesis is twofold, first we will state what performance we expect from the agent on the maze grid and then we will argue what we expect on the bottleneck sub-grid.

Expected time between visits when covering the maze

We know that when one agent tries to cover a similar grid using the Node Counting algorithm without diffusion the standard deviation of the expected time between visits is around 3200, while one full coverage takes between 3200 and 3500 steps.[4] If the agent has not visited a square twice in the last n / 2 number of steps, where n is the amount of steps it takes to cover the grid once, and the agent has taken more than n steps, half of the squares will be visited less than n steps ago and half will be visited more than n steps ago. The expected time between visits will then be n steps.

Since the maze has 320 free squares, one coverage will take at least 320 steps. The expected time between visits and the standard deviation in the expected time between visits are expected to be close to the time it takes to make one coverage, so these expected to be similar to each other and higher than 320 for the Node Counting algorithm without diffusion.

Since the second part of this argument is also valid for the Node Counting algorithm with diffusion, we will expect this algorithm to yield a similar expected time between visits. The dissipated values next to a path an agent takes prevent it from getting turned around as easily as when it is not using diffusion. For that reason, we expect the agent to go up and down the grid more when using this algorithm, which means it leaves recently visited squares earlier and sooner returns to squares that have not been visited in a while. Therefor we expect the standard deviation in the time between visits to be lower when using the Node Counting algorithm with diffusion than when using the Node Counting algorithm without diffusion.

Expected success rates on the bottleneck sub-grid

In order to make a good estimate of how the agent performs on the bottleneck sub-grid using the different algorithms for repeated coverage, we will first look at how it performs when making one visit. By listing all possible moves the agent can make, we will make a probabilistic argument that proves the success rate when one visit is made. Because the agent can only makes the moves listed and will make these moves with the probability that is listed, we can calculate the probability an agent has of making a successful visit by adding the probabilities at which the steps that lead to a successful visit occur together. Then we will assume that the agent performs similarly when it covers the sub-grid repeatedly. By running the simulation we will be able to tell whether this holds true. To only see the results of this small simulation, please skip to 'comparing the success rates'.

Expected visit of both algorithms in one coverage on the bottleneck sub-grid

To ensure that this simulation will be easy to follow, we will start a step by first stating the path the agent has taken to get to a square where a certain decision has to be made. Then we will state the a-priori probability of the agent arriving at that square via the stated path. By stating these marginal probabilities we can concisely compute the success rate an algorithm has on an unvisited bottleneck sub-grid. We do this by simply adding all marginal probabilities that belong to path that result in a successful visit together. We will list all possible paths the agent can take when it enters the bottleneck sub-grid from Grid 1. Since we sum the marginal probabilities for every possible path the agent can take we will in this way prove both algorithms' efficiency in this case.

Success rate of node-counting without diffusion

First, we will show the success rate of the node-counting algorithm without diffusion on an unvisited grid, which means all squares are marked zero. The agent will first arrive at A from grid one, marking Grid 1 exactly one. Then it will randomly visit one of B and C, both with a probability of 0.5.

A-B-D, 0.5: If B is chosen, D will be picked next. From here, the agent will randomly visit either square C or Grid 3, both are equally likely.

A-B-D-Grid 3-D-C-Grid 2, 0.25: If from here Grid 3 is visited, the agent will eventually return to D. Then the only option is to visit C and the Grid 2, making this a successful visit.

A-B-D-C-Grid 2-C, 0.25: If instead C is chosen, Grid 2 will be visited next. Once the agent is back at C, it has two options. It can either retrace its steps and move to D or it can move to A.

A-B-D-C-Grid 2-C-D-Grid 3, 0.125: If the agent decides to go to D it will have to continue to Grid 3, making this a successful run.

A-B-D-C-Grid 2-C-A, 0.125: If A is chosen instead, the agent can return to Grid 1. It can also choose to follow the path B-D-Grid 3, which would make this a successful visit with an a-priori probability of 0.0625.

A-B, 0.5: So, the chance of making a visit where B is chosen is the sum of all earlier stated probabilities of successful visits. The resulting chance is 0.4375.

A-C, 0.5: If C is chosen from A, the agent will then randomly visit Grid 2 or D.

A-C-Grid 2-C-D-Grid 3, 0.25: if Grid 2 is visited, the agent will then visit C, D and Grid 3, making this a successful visit.

A-C-D, 0.25: If D is chosen, the agent will then visit B or Grid 3.

A-C-D-B-A, 0.125: Visiting B leads the agent back to square A. Here it can return to Grid 1, making this an unsuccessful visit, or travel to square B or C.

A-C-D-B-A-B-D-Grid 3-D-C-Grid 2, 0.0417: If the decision made is B, the agent will continue to Grid 3 and Grid 2, resulting in a successful visit.

A-C-D-B-A-C-Grid 2-C-D-Grid 3, 0.0417: A similar result follows when the agent chooses C when the agent visits Grid 2 and Grid 3, making this or a successful visit as well.

A-C-D-Grid 3-D-B-A, 0.125: Traveling to Grid 3 from A-C-D causes the agent to continue to square A. From here it can decide between Grid 1, B and C. Choosing Grid 1 makes this an unsuccessful visit.

A-C-D-Grid 3-D-B-A-B, 0.0417: Reaching B again will cause the agent to either make a successful visit by choosing the path D-C-Grid 2, 0.0208, or return to the previous choice, where picking B is no longer an option, turning this in a successful visit via C-Grid 2, 0.0104 half the time.

A-C-D-Grid 3-D-B-A-C-Grid 2, 0.0417: Choosing C will cause the agent to go to Grid 2 which makes this visit successful.

A-C, 0.5: So, if C is chosen, by adding all success rate we find that a success rate of 0.3959 is achieved.

From combining these initial options of choosing B or C, we can deduce that node-counting without diffusion leads to a success rate of 0.8334 on the first run on the bottleneck grid when taking into account the assumptions made.

Success rate of node-counting with diffusion

Now, we will show the success rate of the node-counting algorithm with diffusion on an unvisited grid, which means all heuristic values are 0. We will assume that the agent will increase the heuristic value of a cell by 1 when leaving it, and will up the heuristic value of the surrounding cells by a factor x, with x < 1.

The agent will again first arrive at A, where it will arrive at a value of x and adds a marked value of 1+x. It will randomly choose between B and C that both still be marked zero, both with the same probability.

A-B-D-Grid 3-D-C-Grid 2, 0.5: If B is chosen, its following path is straightforward. It will have to continue towards D, since C is marked x. After that it will reach Grid 3 for the same reason. Then, it will visit D again, C and then Grid 2, making this a successful run.

A-C, 0.5: If it chooses C from A, the agent then has to choose between Grid 2 and D.

A-C-Grid 2-C-D-Grid 3, 0.25: Choosing Grid 2 ensures that the agent will visit C, D and Grid 3 afterwards, due to the value of x on B, making this visit a successful one.

A-C-D-Grid 3-D-B-A-Grid 1, 0.25: Choosing D will cause the agent to pick Grid 3 next, leading it to the path D-B-A. At this point the agent will return to Grid 1 which is marked 1 + 2*x while B and C are marked 1 + 3*x. The agent does not visit Grid 2 during this visit, making this run unsuccessful.

In total, one of four equally likely option leads to an unsuccessful run, giving the node-counting algorithm with diffusion a success rate of 0.75 on an unvisited grid, when all earlier made assumptions are taken into account.

Comparing the success rates

From the previous analysis, we see that on the bottleneck sub-grid, when starting in Grid 1 on the first run, the Node Counting algorithm without diffusion has a success rate of 0.83, while the Node Counting algorithm with diffusion has a success rate of 0.75. If the coverage of these algorithms is similarly efficient on the particular sub-grids, the Node Counting algorithm without diffusion will perform better on the first coverage of the entire grid.

The hypothesis that results from these expected values is the following. We expect that the Node Counting algorithm without diffusion has a success rate of around 0.83 on repeated coverage when taking the made assumptions into account and we expect that the Node Counting algorithm with diffusion has a success rate of around 0.75 on repeated coverage when taking the made assumptions into account. Therefor we expect that the Node Counting algorithm without diffusion performs better than Node Counting algorithm with diffusion when performing repeated coverage. Furthermore, we expect the Node Counting algorithm without diffusion to have a success rate of around 0.83 when performing repeated coverage, while we expect the Node Counting algorithm with diffusion to have a success rate of 0.75 when performing repeated coverage.

5. Results

Maze coverage average time between visits

In Table 1, the average time between visits of the agent to each square is shown along with the average standard deviation in it. When running the simulation the average time between visits is almost the same for each run. The standard deviation in the time between visits varies between 300 and 330 during the simulation when performing Node Counting with diffusion and between 330 and 360 when performing Node Counting without diffusion.

Table 1 Average time between visits

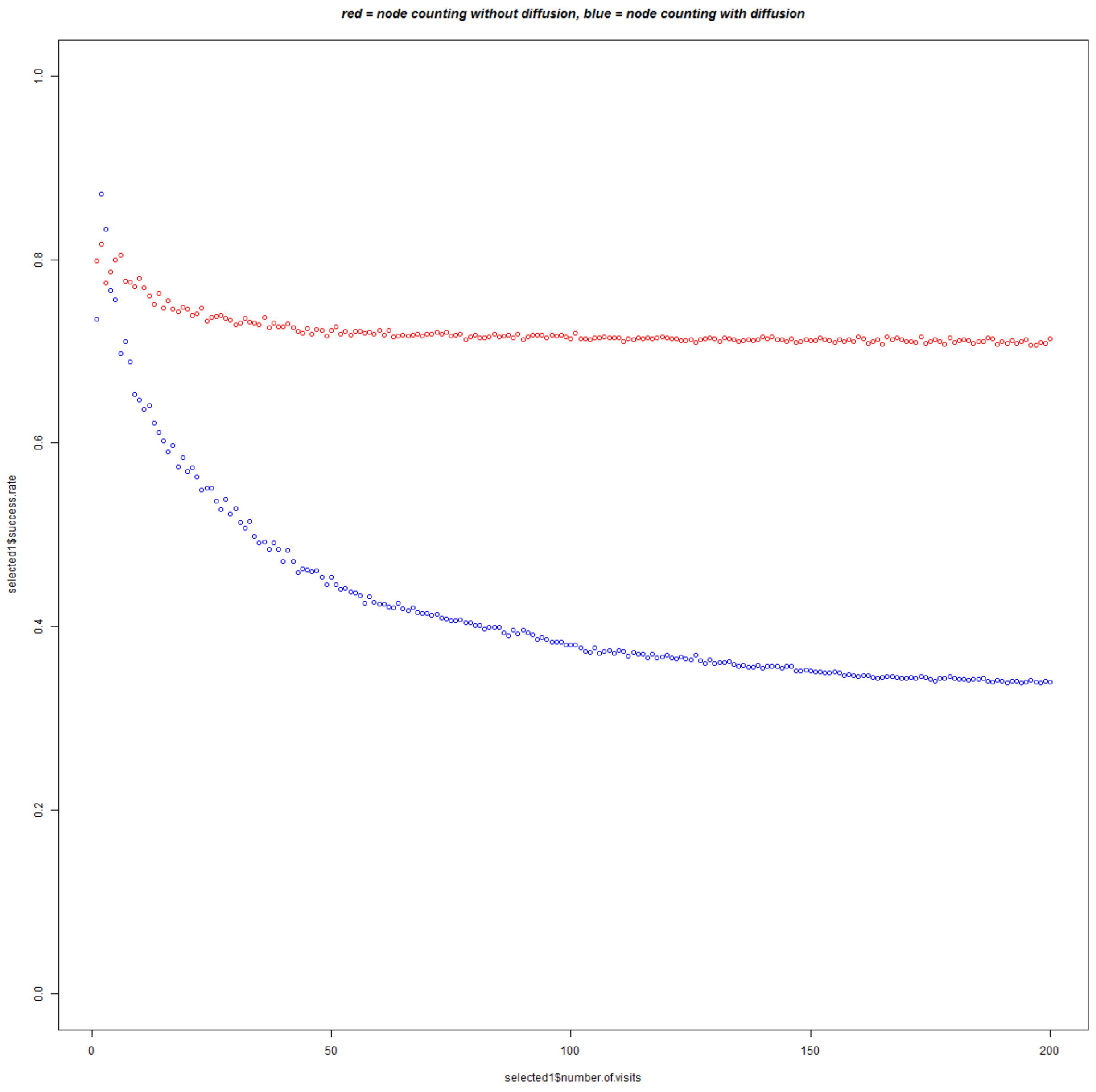
The table below shows average time between visits for both algorithms and the standard deviation in it.

	Node Counting without diffusion	**Node Counting with diffusion**
Average time between visits	**348.0**	**347.6**
Average standard deviation	**337.7**	**308.9**

Bottleneck success rates

When our simulation is run the success rates for when one visit is made converge to 0.82 for Node Counting without diffusion and to 0.75 for the Node Counting algorithm with diffusion. This descends to 0.750 for Node Counting without diffusion and to 0.339 for the Node Counting algorithm with diffusion.

Figure 2: The success rate of the Node Counting algorithm with diffusion and of the Node Counting algorithm without diffusion over the number of coverages.



In this figure, both the success rates of the Node Counting algorithm without diffusion and the success rate of the Node Counting algorithm with diffusion are plotted. The Node Counting algorithm without diffusion shows an initial value of ~0.83 and seems to converge to a value of ~0.711. The Node Counting algorithm without diffusion shows an initial value of ~0.750 and also seems to converge to a value of ~0.339. We can see that both algorithms are not at their peak when one visit is made, but generally perform best when two visits are made.

Table 2 Success rates

The table below shows the success rates for both algorithms and the minimum amount of visits and at an approximation of infinite visits.

Success rate for both algorithms	**Node Counting without diffusion**	**Node Counting with diffusion**
1 visit	**0.82**	**0.75**
200 visits	**0.71**	**0.34**

6. Interpretation

In Table 1, it is shown that when repeatedly covering the maze grid, the average standard deviation of the time between visits is lower when the agent is using the Node Counting algorithm with diffusion than when using the Node Counting algorithm without diffusion. This indicates that when the agent visits a square, the number of steps it has taken since the last time it visited this square is closer to the average when diffusion is not applied.

From Figure 2, we can clearly see that when performing repeated coverage, the Node Counting algorithm without diffusion still has a higher success rate than the Node Counting algorithm with diffusion. The difference found is larger for longer runs.

On the first run, both algorithms are achieving the success rates we predicted for them in the hypothesis section. When making one visit per run, Node Counting without diffusion makes a successful run around 83% of the time, while Node Counting with diffusion does so 75% of the time. The algorithms both behave slightly better when making two visits in a run, Node Counting with diffusion even does so for two through four runs.

We also see that the efficiency both algorithms quickly drops from the values they achieve with when a small amount of visits is made and that the both seem to converge to a value that remains the same when more visits are made. This indicates that at a large enough number of visits, Node Counting without diffusion will achieve an efficiency of 0.71 on the bottleneck sub-grid and Node Counting with diffusion will achieve an efficiency of 0.34 on the bottleneck sub-grid, as can be found in Table 2.

Implications

When looking at these findings keep in mind that many assumptions have been made. The efficiency of the algorithm without diffusion being higher on the bottleneck sub-grid only implies that it performs better if we know that it will perform similarly on the specific sub-grids and when we know that visiting Grid 2 and Grid 3 more often than necessary is preferred over missing them. We also do not know if the agent's performance on the maze grid was much impacted by the shape of the grid when using one or both algorithms.

There are no indications how the efficiency will be if the sub-grids are somehow connected and the agent cannot detect this when covering the grid.

One thing we see in the results is that Node Counting without diffusion performs a lot better than Node Counting with diffusion on covering the bottleneck. When covering this sub-grid repeatedly, the percentage of successful runs the algorithm without diffusion makes is twice that of the algorithm with diffusion.

7. Related work

In “Evaluating Failure in Terrain Coverage by Autonomous Agents”[7] an algorithm named Alarm is introduced. An algorithm that works much like Node Counting, adding pheromone instead of visits, with the difference being that instead of always choosing the neighboring square that has the lowest amount of pheromone, the agent chooses a random neighboring square while the squares with less pheromone have a higher chance of being chosen. The pheromone also evaporates over time, making this both a spatial and a temporal coverage algorithm. This algorithm is used in “Terrain Coverage Ant Algorithms: The Random Kick Effect"[2], where the suggestion is made to also add dissipation to the attributes of the pheromone. This would bring the way the pheromone behaves in simulation closer to the principle it is based on. When dissipation of the pheromone to bordering cells is added to the alarm algorithm, it behaves very much like the Node Counting algorithm with diffusion used in this research. It is not yet investigated if dissipation of the pheromone makes Alarm perform better at coverage.

8. Discussion

There are some points that have to be noted in order to correctly make use of the results of this paper. Some factors assumed in this research may very well be different when investigating this subject in a different context.

One point is that the algorithm with diffusion may perform much better or worse than the algorithm without diffusion at covering the three sub-grids. If one of the algorithms has more difficulty than the other at fully covering each sub-grid before returning to the bottleneck, this may, based on which algorithm does better, amplify or negate the results obtained in this research. It is possible that a difference in performance arises at grids of certain shapes or sizes.

Generally, diffusion forms a circular shape around the point of origin, forming a gradient with its peak at that point. In this research, this effect is simplified by giving the original square a certain value and then giving all squares that are adjacent horizontally and vertically a smaller value. This way, diagonally adjacent squares are ignored, while an agent on an adjacent square cannot see where the diffusion comes from based on the value on that square alone. Also, values added via diffusion that encounter a wall might bounce back instead of disappearing. Accounting for these factors might induce significantly better approximations of how diffusion compares to no diffusion in recurrent coverage.

In the simulation, the assumption is made that the way the agent covers the sub-grids does not affect the way the agent covers the bottleneck. While this holds true on the short term, the shapes of the sub-grids will influence the marked value on squares in the bottleneck after a number of visits.

9. Conclusion

As was expected, the average time between visits is very similar for both algorithms, this does not give much information with which the algorithms can be judged. Also as expected, the average standard deviation in the time between visits is lower when the agent uses Node counting with diffusion. This means that the agent does perform more consistently when using this algorithm than when using Node Counting without diffusion.

In the hypothesis we stated that we expected the Node Counting algorithm with diffusion to perform better on the bottleneck than the Node Counting algorithm with diffusion. We also expected the Node Counting algorithm with diffusion to have a success rate of 0.83 and we expected the Node Counting algorithm with diffusion to have a success rate of 0.75. From the results, we can clearly see that Node Counting without diffusion does indeed on average make more successful visits per total visits when covering the bottleneck sub-grid.

It is also clear that both algorithms perform worse in the long run than they do at their initial visit. Node Counting without diffusion goes down from 0.83 to 0.71 and Node Counting with diffusion goes down from 0.75 to 0.34. Because of these changes in success rates, the difference in performance between both algorithms gets much bigger for multiple visits than it is for one visit.

Answer to research question

The research question posed was "What are the effects of diffusion of pheromone in node counting on recurrence on a grid?": The effects of diffusion on recurrence on a bottleneck sub-grid seem to be negative. This much is clear when looking at the results of the simulation. Node Counting with diffusion performs much worse than Node Counting without diffusion when it is used by a single agent to cover the bottleneck sub-grid. The task it performs worse at is continuously visiting both smaller sub-grids before returning to the main sub-grid. On the other hand, Node Counting with diffusion does visit squares more regularly when covering a maze than Node Counting without diffusion. The effects of diffusion in Node Counting seem to benefit regular visits to a generic well connected grid, but seem to be counter effective when different parts of the grid are connected only through a small pathway. The shape of the grid needs to be taken into consideration before deciding on whether to use diffusion.

Further research

Insects that convey information via pheromones seem to make use of hill climbing based on pheromone levels when going to or shying away from a location another insect has already been at. It may be interesting to look into how algorithms that use diffusion perform compared to non-diffusion counterparts on differently shaped grids and when making use of multiple agents. Further information on how an agent using the Node Counting algorithm with diffusion performs at coverage in different environments would also help put this research in perspective by validating or debunking the assumption that both algorithms used in this research perform similarly in those situations.

10. References

[1] Adler, F. & Gordon, D., 1992. Information collection and spread by networks of patrolling ants. *The American Naturalist,* pp. 373-400.

[2] Dervisi, M., Efremides, O. & Iracleous, P., 2013. Terrain Coverage Ant Algorithms: The Random Kick Effect. *International Journal of Advanced Computer Science and Applications.*

[3] Koenig, S. & Liu, Y., 2001. *Terrain coverage with ant robots: a simulation study,* s.l.: The fifth international conference on Autonomous agents.

[4] Koenig, S., Szymanski, B. & Liu, Y., 2001. *Efficient & inefficient ant coverage,* s.l.: Annals of MAthematics and Artificial Intelligence.

[5] Mason, J. & Menezes, R., 2008. *Autonomous Algorithms for Terrain Coverage Metrics, Classification and Evaluation,* s.l.: IEEE Congress on Evolutionary Computation.

[6] Neumann, J. v., 1966. *Theory of Self-Reproducing Automata (edited and completed by Arthur Burks),* Illinois: University of Illinois Press.

[7] Treverton, R. & Menezes, R., 2009. Evaluation Failure in Terrain Coverage by Autonomous Agents. *IEEE.*

11. Appendix

The simulation script netfile.nlogo and representation script rep.R used are available at https://drive.google.com/open?id=0Bxx0vueFj5DSbVNOOHhnVkxsTlk. The simulation is run in Netlogo 4.1.3 and the representation is made using R 3.2.3, these programs are necessary for running the scripts. The way these simulations work is explained in the Simulation section under Methods. In order to run the experiment, open netfile.net using Netlogo. Now the simulation process is visualized by clicking "Setup" and then "Run". Ctrl+shift+B opens the BehaviorSpace. Running "experiment" will execute the bottleneck part of the experiment and running "mazeexperiment" executes the maze part of the experiment. Be sure to check "table output". To find the output from the maze experiment, simply open the resulting table in any text editor. To visualize the output from the bottleneck experiment open the script rep.R using R. Click "file" and then "change dir…" and choose the directory the experiment is saved in. Click on any place within the rep.R file and enter "crtl+A" and "crtl+R" on the keyboard to output the visualization to the directory the table is in.