The University of Birmingham
School of Computer Science
MSc in Advanced Computer Science

Behvaiour of Complex Systems

Termite Mound Simulator

John S. Montgomery
`msc37jxm@cs.bham.ac.uk`
Lecturer: Dr L. Jankovic

May 6, 2004

Contents

List of Figures

Chapter 1

Introduction

1.1 Real Termites

Real (worker) termites (Macrotermes Bellicosus) are small in size, completely blind and wingless - yet they have been known to build mounds 30 metres in diameter and several metres high [2]. It is generally accepted that a process referred to as stigmergy, by Pierre-Paul Grasse, is responsible for allowing such feats of (insect) engineering and it works in this fashion [1]:

1. The termites move at random dropping pellets of saliva and earth on raised ground to form small heaps.
2. The small heaps encourage other termites to join in and the biggest heaps develop into columns.
3. As the columns are built, nearby ones will tend to be built towards each other.

Figure 1.1 shows a real termite mound, demonstrating the kind of scale that these termite built structures can achieve.

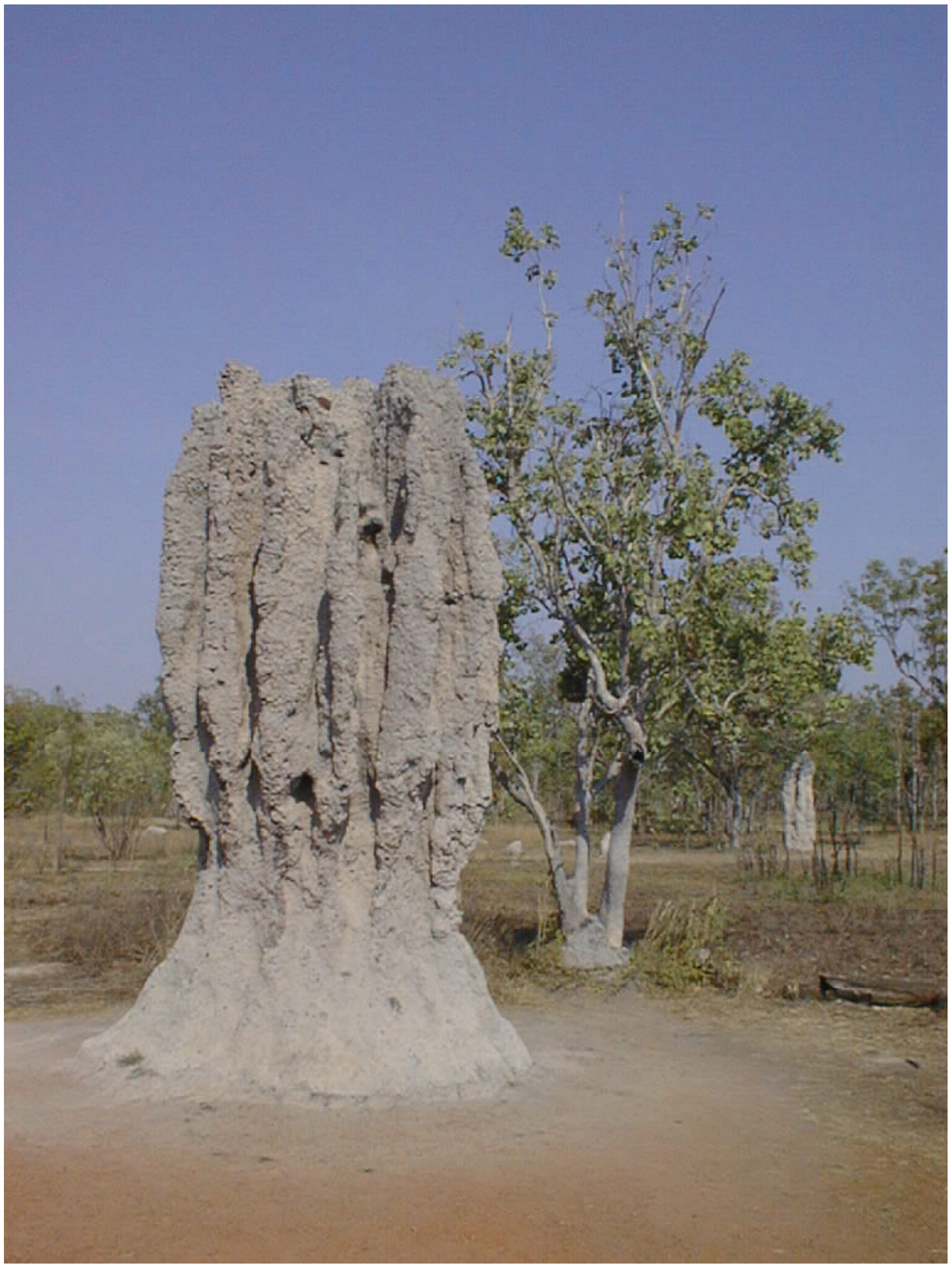


Figure 1.1: Magnetic Termite Mound, Litchfield National Park (http://www.travel-library.com/pacific/australia/nt/venkat/)

Chapter 2

Technical Specification

2.1 Physics

The simulation is run using a 3D grid, 128x128x256 elements in size. Each grid element is either empty or full. The termite agents in the simulation or able to move about this grid, with a few caveats:

1. "Unsupported" termites fall under the influence of gravity, until they become supported again (See Figure 2.1).

2. If there are no filled grid elements next to a termite it is considered unsupported.

3. A termite may choose any empty neighbouring grid positions (26 maximum) to move into.

4. Elements "outside" the grid are considered to be full (thus blocking the termites), except for elements beyond the top the grid (gravity ensures the termites do not escape beyond that).

5. If a termite happens to find itself "inside" a full grid element then it is moved upwards (against gravity) to the nearest empty element.

6. Any number of termites may share same grid location.

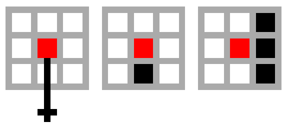


Figure 2.1: An unsupported Termite (Red) will fall towards the ground.

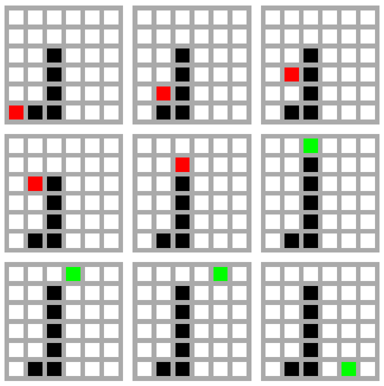


Figure 2.2: Termite Building (Red), then Foraging (Green)

This provides for a simple qualitative physics model. The termites are able to climb over the surface of any objects in the grid - they can even climb upside down. The moment the move away from the surface too far they will be affected by gravity and will fall to the ground.

2.2 Pheremones

When a termite decides to build on a grid element (thus making it full) it also drops some pheremone around the local area. The pheremone is added to an underlying "Pheremone Grid" with a certain radius of the termite. The exact amount of pheremone added to the nearby grid elements is inversely proportional the distance from the element to the termite. The highest concentration of pheremone added is at the termites (current) location.

2.3 Simulated Termite Behaviour

Simmulated termites have two basic behaviour states:

1. Building.
2. Foraging.

See Figure 2.2 for an example of this process in action.

2.3.1 Building

Whilst building termites will follow height and pheremone gradients. In other words they will tend to climb higher and move towards concentrations of pheremone. Exactly how much more important height or pheremones are to the termites decisions is controlled by a set of weights. These weights are combined with the raw values of the change in height and pheremone levels to compute a "score" for a given grid element. However grid elements that would leave the termite "unsupported" are given a score of zero, so termites do not try to jump into the air uneccesarily. Once the scores for the respective grid elements (including the one the termite currently occupies) have been computed then the termite chooses which one to move into next. In a similar way to the ants choosing a path in "Ant Algorithms" [3] the termites have a "greediness" parameter. The termites greediness controls how likely the termite is to simple choose the "best" move. If the best move is not chosen then a move will be chosen using a roulette wheel/proportional seclection method. The greediness factor in effect controls how much the termite will tend to meander slightly off the "best" route. If at any point the termite chooses its current location (i.e. it chooses not to move) then this is assumed to mean a local maximum has been reached. The termite will then randomly decided whether to build at that location (or to wait till later). If the termite does decided to build at that location, then it's behaviour changes to "foraging" afterwards.

2.3.2 Foraging

Foraging termites simple move randomly, including off the edges of drops, until they are back on solid ground (i.e. back on the very bottom of the grid). They are then assumed to pick up more building material and so start "building" again.

Chapter 3

Design

The design roughly follows the "model, view, controller" paradigm. So each class can generally be categorised based on where it falls in the MVC approach. For specific details see the java api documentation supplied and/or the source code.

3.1 Model

The model classes are:

- Termite
- TermiteSim

They are the core of the simulation. They control what is actually happening, with respect to the termites building mounds, following pheremones etc.

3.2 View

The view classes are:

- TermiteView
- SplitViews
- Mound3DView

These classes merely present to the user representations of the "model" (i.e. the simulation). In particular the TermiteView class is used to give the user two-dimensional views, either side-on or top-down and the Mound3DView is used to provide a 3D snapshot of the current state of the simulation.

3.3 Controller

The controller classes represent the “wiring” between the view and the model. They are the classes that mediate user interaction with the model.

- Termites
- TermiteApplet

The Termites class is merely responsible for creating a window, into which the the main program will run. The TermiteApplet class is the main “controller”. It sets up the various views and provides interface controls that let the user interact with those views and also run simulations.

Chapter 4

Operation

4.1 Basics

To run the termite simulation from an executable jar file, either double-click on it to launch java or use the -jar command:

```
java -jar termites.jar
```

Otherwise if one compiles the source code, the "main" class is Termites, so:

```
java Termites
```

Should suffice to start the simulation.

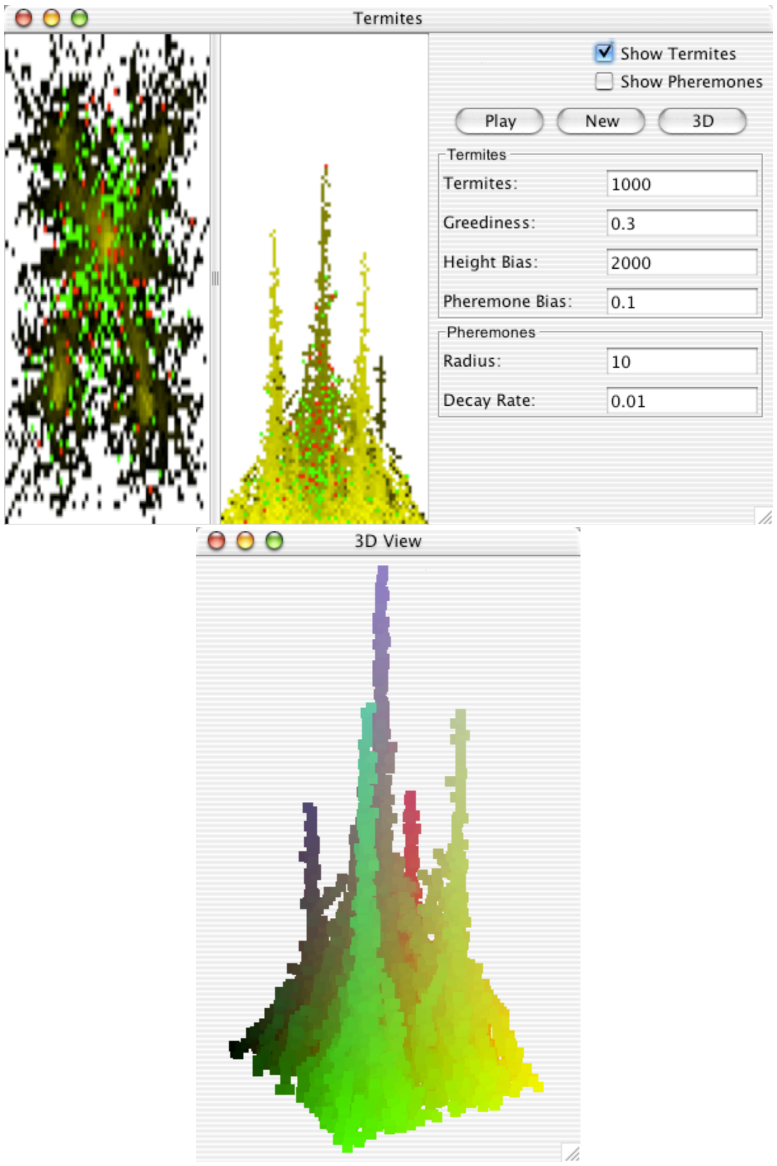
Figure 4.1 shows the application running. The top screen-shot shows the main application window. The two dimensional views use a simple colour scheme to convey depth. The darker pixels represent structure that is farther from the view, the lighter (towards yellow) pixels represent those that are closer. The termites are drawn [1] on the structures as single red (building) or green (foraging) pixels.

The key controls are the three buttons:

- Play/Pause
- New
- 3D

They are all pretty self-explanatory. Play/Pause is used to pause the simulation and to continue running it. The text on the button changes, depending on whether the simulation is currently paused or not. New creates a new simulation, effectively resetting everything. It uses the current settings to make the new simulation. 3D makes the 3D view appear, which can be used to see a snapshot of the current built structure.

[1] A z-buffer is used, so termites may become obscured from view by intervening structure



[✓ Show Termites]
Show Pheremones

Termites

Termites:	1000
Greediness:	0.3
Height Bias:	2000
Pheremone Bias:	0.1

Pheremones

Radius:	10
Decay Rate:	0.01

Figure 4.1: Main and 3D Views

4.2 Termite Parameters

The termite parameters are as follows:

- Termites - how many termites will appear in the simulation.
- Greediness - a probability specifying how likely the termites will always take the "best" course of action.
- Height Bias - a weight controlling how much the termites favour climbing.
- Pheremone Bias - a weight controlling how much the termites favour following pheremones.

With the exception of the "Termites" parameter, all of these settings can be changed on the fly. Simply edit the value in the text field and then press enter to make the value update in the simulation. The number of termites in a simulation cannot be altered once a simulation is running, so the value of the "Termites" parameter is only used when a new simulation is started.

4.3 Pheremone Parameters

The pheremone parameters are:

- Radius - how far the pheremones extend.
- Decay Rate - the proportion of the pheremones that are removed at every time-step.

Both of these parameters can be altered on the fly. As before, edit the values and then press enter to make them stick. You can see the effects of the pheremones by ticking the "Show Pheremones" check box.

4.4 3D View

The 3D view creates a snapshot model of the termite mound. The model is rendered using an isometric style, so no foreshortening occurs and can be rotated, around the z-axis, by clicking then dragging the mouse horizontally. It is also possible to "zoom" in and out, by pressing the I and O keys.

Chapter 5

Sample Results

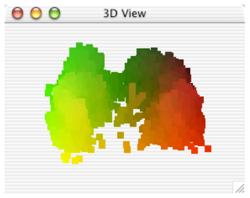
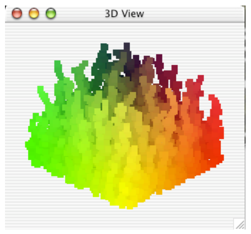
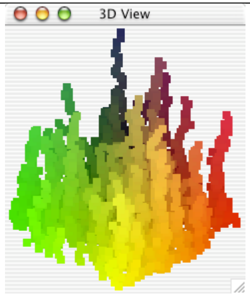
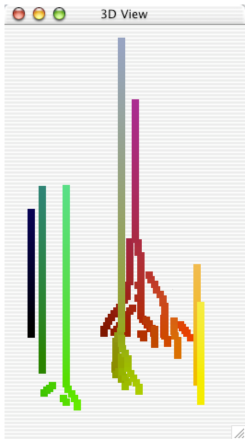
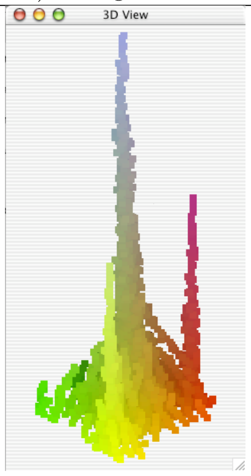
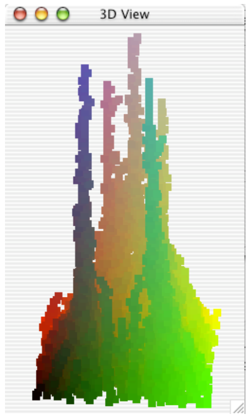
 a) low height bias	 b) zero greediness	 c) no pheremones
 d) maximum greediness	 e) default parameters	 f) lots of termites

Table 5.1: Termite Mounds

The mounds shown in Table 5.1 were created using various different parameters. In particular mounds b) and c) demonstrate quite chaotic structures, with spires all over the place. Whereas d) shows a highly regular shape - mainly perfectly straight tall spires and overly ordered.

In terms of how similar to real termite mounds (Figure 1.1) they look f) is probably the best. This is because using a lot more termites (1000 as opposed to the usual 100) allows several competing spires to be built close together, in effect forming a wall. Given the likely numbers of termites in a real colony, one would imagine that adding more virtual termites to the simulation may result in increased mimicry of real termite mound shapes. However only up to a point, as the termite behaviour modelled is probably much simpler than that exhibited by real termites.

Chapter 6

Extensions and Improvements

There are a couple of cosmetic improvements that could be made. The main one being to make the main view be 3D (rather than only providing a separate static view). This would make life a lot easier in terms of understanding what is going on. If done right it would allow the user to zoom in and possibly even track a single termites progress. To achieve this kind of interactivity, but still maintain sensible performance would require a serious amount of work and/or the use of 3D acceleration, e.g. via Java3D or an OpenGL binding for Java.

As it stands the pheremone model is very simplistic. Currently the pheremones are laid down in a static pattern and merely decay over time. A better approach would be to allow the pheremones to spread slowly over time. The weight of the pheremone molecules could then be a user alterable parameter, changing whether pheremones will tend to fall downwards. If the pheremones spread downwards over the structure this would alter the behaviour of the termites a lot, as they would become more likely (or less) likely to build around the bases of spires.

Another extension would be to apply some sort of CA-based structural component to building. It may be possible to create a simple cell-based model of the stresses and strains of the built structure. Then perhaps even simulate behaviour that would represent collapse and slides, altering the final shapes. Also coupled with information concerning the underlying stresses and strains it could be possible to apply simple genetic algorithms or other optimisation techniques to find the "perfect" set of parameters for building a termite mound.

References

[1] Collective intelligence in social insects.
http://ai-depot.com/Essay/SocialInsects.html.

[2] Macrotermes bellicosus: Special inspecta investigation.
http://www.insecta-inspecta.com/termites/macrotermes/.

[3] L.M. Gambardella M. Dorigo, G. Di Caro. Ant algorithms for discrete optimization. *Artificial Life*, 5(2), 1999. Special Stigmergy Issue.

Appendix A

Simulation Application and Source Code

The source code and compiled executable versions of the simulator will be made available via the authors School of Computer Science personal website at:

`http://studentweb.cs.bham.ac.uk/~msc37jxm/complex-systems/`