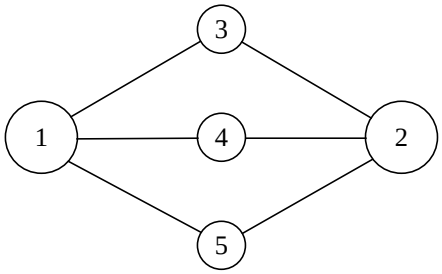
It is also interesting to note that when initialising the network with a large number of time steps, a combination shortcut / blocking problem occurs. The ABC system has ‘overlearnt’ one particular route, and when it becomes congested, it does not find the other route. Adding noise however, largely overcomes this problem. (Compare the last two numbers).

Other experiments were performed on the network of Figure 26. Here I expected the fixed routing scheme to perform worse. The capacities of nodes 3, 4 and 5 are now 30 calls each, and again calls are only made from node 1 to 2 or from 2 to 1. The other settings remained the same.

FIGURE 26. Test network 2



All experiments have been performed five times with different call patterns for every experiment. The results were this time:

- Fixed paths: 15.0%
- Mobile agents: 4.2%
- Ants (0% noise, initialisation period of 50): 8.4%
- Ants (75% noise, initialisation period of 50): 6.9%
- Ants (25% noise, initialisation period of 50): 8.2%
- Ants (0% noise, initialisation period of 5000): 15.1%
- Ants (10% noise, initialisation period of 5000): 8.2%

Because of their rapid adaptations, the mobile agents performed best again. The fixed routing tables performed poorly as expected, because one of the three nodes was not used at all. When observing the simulations of the ants with 0% noise and 5000 time steps of initialisation, I saw that in four of the five performed experiments one of the nodes remained unused here as well. Ants only kept on choosing between two routes, as these routes were ‘where the pheromone was’. Noise again overcame this problem. With these two networks one could see during the simulations that bi-directionality of routes in the ABC system lead to some performance problems.

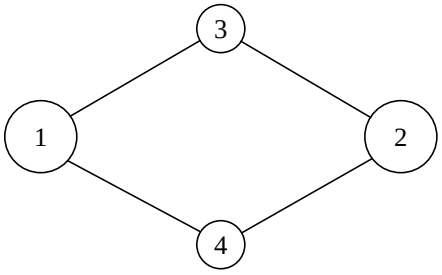
One can conclude that a least cost strategy like the mobile agents method is best for simple topologies. However, when the topology becomes more complex (like the one of Figure 13 on page 37), the ants method will clearly lead to better load balancing.

Appendix D More experiments

Although complex networks might be more interesting, I also did some experiments with small networks. These experiments are not complete enough to make clear statistically grounded statements about their results, but I append them to this thesis because they might provide some useful insights in the behaviours of the different load balancing strategies.

Consider network 1 in Figure 25. Calls are only generated from node 1 to node 2 or from node 2 to node 1, with the same probability each. An optimal fixed routing strategy might be to route all traffic from node 1 to node 2 via node 3, and from node 2 to node 1 via node 4. With this fixed routing strategy is tested, as well as with the mobile agents (that maximise the minimum spare capacity on the route) and several ant conditions. The parameter settings are generally the same as the ones from Section 5.1, but the parameters of the agents were adapted to the size of the network. Nodes 1 and 2 have capacity 200, and node 3 and 4 have capacity 40. On average one call in 2 time steps of length 160 is generated, so that on average 80 calls will be present on the network, exactly the number of calls that can be placed between nodes 1 and 2 if their routes are evenly spread over nodes 3 and 4.

FIGURE 25. Test network 1



Call failures were monitored between time steps 2000 and 4000. The percentages are averages over five experiments for each ant-condition, and ten for the other two conditions, and the call sequences were different for every experiment. The results were:

- Fixed paths: 5.6%
- Mobile agents: 5.9%
- Ants (0% noise, initialisation period of 50): 7.4%
- Ants (75% noise, initialisation period of 50): 8.3%
- Ants (25% noise, initialisation period of 50): 8.3%
- Ants (0% noise, initialisation period of 5000): 27.1%
- Ants (10% noise, initialisation period of 5000): 6.9%

The topology is much simpler and does not have to be ‘learnt’ here. Therefore the mobile agents perform very good because of their rapid adaptation. The fixed tables turned out to be approximately as good as the mobile agents. Probably an adaptive routing scheme would not be very beneficial for these particular statistics, unless it is reacting very fast.

(We would have concluded a difference if we would have chosen a 0.05 significance level)

Ants 5% noise
118 118 136 186 211 129 225 125 148 99 (avg 149.5 dev 40.6872)
7420 7346 7377 7343 7252 7413 7295 7317 7359 7364 (avg 7348.6 dev 48.6193)
(dropped calls avg 1.99% dev 0.54%)

Ants 5% noise + changing call probabilities
115 162 319 336 131 238 195 179 87 162 (avg 192.4 dev 78.5063)
7349 7351 7210 7127 7411 7282 7247 7328 7376 7376 (avg 7305.7 dev 83.962)
(dropped calls avg 2.56% dev 1.05%)

Ants 5% noise, stopped from being launched
146 144 153 269 268 157 251 165 171 133 (avg 185.7 dev 51.5811)
7392 7320 7360 7260 7195 7385 7269 7277 7336 7330 (avg 7312.4 dev 58.7047)
(dropped calls avg 2.48% dev 0.69%)

Ants 5% noise, stopped from being launched + changing call probabilities
108 279 645 654 213 272 276 240 263 324 (avg 327.4 dev 169.886)
7356 7234 6884 6809 7329 7248 7166 7267 7200 7214 (avg 7170.7 dev 171.59)
(dropped calls avg 4.37% dev 2.27%)

C.1 Student's t-test

A generally accepted and robust way to compare statistical data is Student's t-test. A 2-tailed test has to be performed on the data, because we do not know in advance which of the two conditions will probably be better than the other. The significance level chosen is 0.01. This means that if there is a probability of 0.01 or less that the observed results were obtained by chance, given that the two compared distributions have equal means, we will decide that the distributions are in fact different. The procedure is as follows [Robson 73]:

- Let d be the difference between a pair of scores. The sum of the differences of all pairs is Σd.
- Sum also the squared differences between each pair of scores - Σd^2
- The obtained value for t is:

$$t = \frac{\sum d}{n} \div \left(\sqrt{\frac{\sum d^2 - (\sum d)^2 / n}{n(n-1)}} \right)$$, where n is the number of samples.

- The test has n-1 degrees of freedom, which is 9 for this experiment. The value to compare t with can be found in any table of the so-called t-distribution. For 2-tailed experiments with 9 degrees of freedom, and a 0.01 significance interval, this value is 3.250.
- If t is outside the interval [-3.250, 3.250], there is only a 0.01 probability that the observed results were obtained by chance. We therefore conclude that the condition that resulted in a smaller average number of call failures is better than the other.

The conclusions drawn in Section 5.3 are based on this test. For example, comparing the condition of ants with 5% noise, with ants with 5% noise and changing call probabilities, results in a value for t of -2.66. Therefore we cannot conclude at the 0.01 significance level that these situations differ in performance.

272 367 434 419 268 356 353 302 225 313 (avg 330.9 dev 63.7251)
7192 7146 7095 7044 7274 7164 7089 7205 7238 7225 (avg 7167.2 dev 69.8352)
(dropped calls avg 4.41% dev 0.85%)

Mobile agents new + stopped from being launched
817 401 707 483 332 559 394 446 427 250 (avg 481.6 dev 162.397)
6721 7063 6806 7046 7131 6983 7126 6996 7080 7213 (avg 7016.5 dev 142.811)
(dropped calls avg 6.42% dev 2.17%)

Mobile agents new + changing call probabilities + stopped from being launched
600 610 1178 723 371 615 461 461 558 447 (avg 602.4 dev 215.809)
6864 6903 6351 6740 7171 6905 6981 7046 6905 7091 (avg 6895.7 dev 215.863)
(dropped calls avg 8.03% dev 2.88%)

Ants 0% noise
122 98 149 161 201 100 204 102 111 91 (avg 133.9 dev 40.4239)
7416 7366 7364 7368 7262 7442 7316 7340 7396 7372 (avg 7364.2 dev 48.1452)
(dropped calls avg 1.79% dev 0.54%)

Ants 0% noise + changing call probabilities
110 166 369 372 120 225 212 196 96 172 (avg 203.8 dev 92.7586)
7354 7347 7160 7091 7422 7295 7230 7311 7367 7366 (avg 7294.3 dev 98.4155)
(dropped calls avg 2.72% dev 1.24%)

Ants 0% noise + stopped from being launched
117 117 136 212 233 114 228 143 160 125 (avg 158.5 dev 45.3194)
7421 7347 7377 7317 7230 7428 7292 7299 7347 7338 (avg 7339.6 dev 56.9108)
(dropped calls avg 2.11% dev 0.60%)

Ants 0% noise + changing call probabilities + stopped from being launched
101 277 602 615 203 258 293 252 320 295 (avg 321.6 dev 154.869)
7363 7236 6927 6848 7339 7262 7149 7255 7143 7243 (avg 7176.5 dev 159.582)
(dropped calls avg 4.29% dev 2.06%)

Ants 2% noise
111 95 132 189 203 118 201 117 137 89 (avg 139.2 dev 40.7892)
7427 7369 7381 7340 7260 7424 7319 7325 7370 7374 (avg 7358.9 dev 47.6413)
(dropped calls avg 1.86% dev 0.54%)

Ants 2% noise + changing call probabilities
107 156 346 374 133 235 197 183 97 167 (avg 199.5 dev 89.3781)
7357 7357 7183 7089 7409 7285 7245 7324 7366 7371 (avg 7298.6 dev 94.7589)
(dropped call avg 2.66% dev 1.19%)

Ants 2% noise, stopped from being launched
135 106 166 245 233 154 224 155 138 142 (avg 169.8 dev 44.8727)
7403 7358 7347 7284 7230 7388 7296 7287 7369 7321 (avg 7328.3 dev 51.4394)
(dropped calls avg 2.26% dev 0.60%)

Ants 2% noise, stopped from being launched + changing call probabilities
104 268 654 631 219 286 259 305 272 319 (avg 331.7 dev 165.492)
7360 7245 6875 6832 7323 7234 7183 7202 7191 7219 (avg 7166.4 dev 165.652)
(dropped calls avg 4.42% dev 2.21%)

Appendix C Details on the results

The results shown in Section 5.3 only showed the averages and standard deviations of the number of call failures during time steps 7500 - 15000. The numbers of call failures for each individual experiment are given in this appendix.

For each condition ten times the numbers of call failures are given in the first line, and the number of successful calls in the second. The averages and standard deviations are shown between brackets.

The numbers are given in the order of the experiments. Recall the experimental design in Section 5.3: The call patterns and call probabilities for run *x* in the situations with fixed call probabilities correspond to the call patterns and call probabilities for run *x*–1 in the situations with changing call probabilities (except for the first experiment with unchanging call probabilities - this situation corresponds to the 10th experiment with changing call probabilities). In this way, for pair-wise comparisons between two cases of unchanging call probabilities, or between two cases of changing call probabilities, one just need to compare the *x*-th number of the one, with the *x*-th number of the other. But for pair-wise comparison between a condition with unchanging call probabilities, and one with changing call probabilities: Compare the *x*-th number of the situation with unchanging call probabilities with number *x*–1 for the situation with changing call probabilities (except when *x*=1, then you have to perform a pair-wise comparison with the 10th number of the experiments with the unchanging call probabilities).

Without load balancing (first line: number of call failures, second line: number of successful calls):
877 902 816 1351 1122 777 877 943 916 845 (avg 942.6 dev 162.146)
6661 6562 6697 6178 6341 6765 6643 6499 6591 6618 (avg 6555.5 dev 167.217)
(dropped calls avg 12.57% dev 2.16%)

Without load balancing + changing call probabilities
905 790 1312 1126 802 875 928 936 832 890 (avg 939.6 dev 153.067)
6559 6723 6217 6337 6740 6645 6514 6571 6631 6648 (avg 6558.5 dev 157.662)
(dropped calls avg 12.53% dev 2.04%)

Mobile agents
733 583 760 773 724 689 700 627 672 631 (avg 689.2 dev 58.4051)
6805 6881 6753 6756 6739 6853 6820 6815 6835 6832 (avg 6808.9 dev 43.9806)
(dropped calls avg 9.19% dev 0.78%)

Mobile agents + changing call probabilities
635 727 800 688 729 722 632 714 578 706 (avg 693.1 dev 59.7385)
6829 6786 6729 6775 6813 6798 6810 6793 6885 6832 (avg 6805.0 dev 38.7608)
(dropped calls avg 9.24% dev 0.80%)

Mobile agents new
330 266 318 409 399 281 335 341 268 214 (avg 316.1 dev 57.582)
7208 7198 7195 7120 7064 7261 7185 7101 7239 7249 (avg 7182.0 dev 62.7838)
(dropped calls avg 4.22% dev 0.77%)

Mobile agents new + changing call probabilities

FIGURE 24. The simulations are controlled via this dialog box.

The simulations will run as follows. The first half (in the example of Figure 24 the first ten) of the experiments will run the experiment with load balancing *on*, during the whole period of the simulations. The second half (the last ten) will run it with load balancing *on*, until the monitoring starts, and at that moment the simulation will turn its load balancing *off*.

Figure 23 shows a snap shot of the simulation program. The main window, labelled ‘ABC Network Simulation Tool’, contains the view of the network. The colours indicate the load of the nodes, according to a traffic light model (green-yellow-red - the more close to red, the heavier the load on the node).

The left hand side of this window provides some data of the current situation, and some action buttons to control the simulation. The time is shown, together with the number of successful and dropped calls so far, and the number of calls that is currently on the network. The simulation can be started, stopped, resetted, or run step-by-step, and one can choose to highlight a specific route between two nodes (here it is the route from node 25 to node 1). One can also choose to display the mobile agents if they are used for load balancing, and start to monitor traffic to a given file, specified with a dialog box.

The windows labelled ‘Node Interface’ appear when clicking on a corresponding node, and give information about this node. The displayed network is populated by ants, so a possibility is to show a specific pheromone table via a node interface. The figure shows node 25’s pheromone table for node 1. It shows that in this situation ants on node 25 that want to go to node 1 have 0.58 probability of going next to node 24, a 0.31 probability of going next to node 26, and 0.11 of going via node 30.

The window labelled ‘NodeWithAnts’, on the bottom left corner, is a special ‘inspect’-window that will appear when selecting the ‘inspect node’ button on the node interface for node 14. One can select all datafields for that node, for example its present call records (which are displayed now), or its pheromone tables.

The top four windows on the left are also inspect-windows. The upper one displays the contents of the network interface object. When selecting the field **network** with the middle mouse button, one can inspect the network. Then **callGenerator** was selected, and then **randomNodeNrStream**, displaying the call probability array field in its window.

The Workspace-window is a VisualWorks command tool, and is used here to start the program.

B.0.1 Running simulations

For running repeated simulations like the ones described in Section 5.3, a dialog box is made to control the different conditions. It is depicted in Figure 24, and started by selecting the ‘file’-menu, item ‘Simulation - Settings’ in the main window.

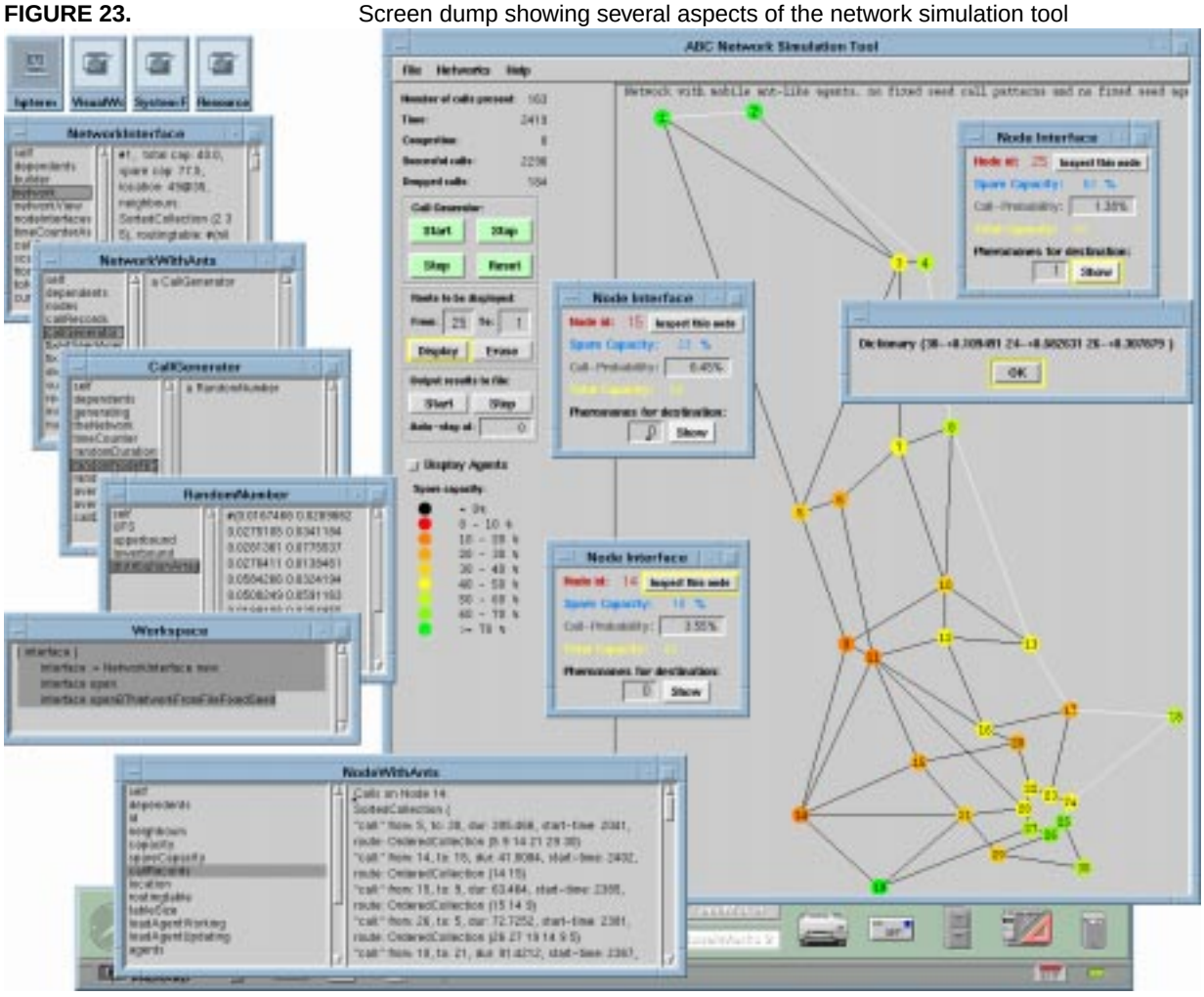
Appendix B The Network Simulation Tool

The network simulation tool written for the test and evaluation of the distributed load balancing techniques supports numerous features. It is highly integrated in the VisualWorks environment. This integration can be very beneficial:

- Tools like object inspectors can be called during the simulation, so that every part of the simulation model (and also other components of the program) can be inspected at any moment.
- Next to the few parameter settings indicated by the Simulation - Settings dialog box (Figure 24); one could easily adapt other parameter settings, even during run-time of the simulation.

Note that these properties are ‘extra’, and one can only make full use of them if one has experience with the Smalltalk / VisualWorks environment. Otherwise one uses only the functionality that is directly built into the program.

FIGURE 23. Screen dump showing several aspects of the network simulation tool



- display mobile agents on the network;
- highlight a particular route, and erase the highlighting.

The outline of the window and menu-bars are specified with the VisualWorks canvas editors.

NodeInterface. When clicking on one of the network nodes, a window appears with information on that node. The node interface has aspect adaptors on the nodes spare capacity and its call probability. Its call probability can also be changed via the node interface. It also contains methods that have to be called when pressing one of the action buttons. These buttons allow the user to launch parent- and load agents by hand, and to open an Inspect-window of the node. The inspect window allows you to inspect every detail of the node, e.g. its routing table or its current call records.

NodeWithAntsInterface. This window is adapted for nodes that populate ants. There are no buttons to launch agents anymore; now the pheromone table for one particular other node can be inspected. Three of those windows are shown in Figure 23 on page 73.

SimulationDialog. This class represents dialog box can be opened via the main window and allows the user to change detailed settings about the simulation, and to perform repeated simulations. Methods are written with the purpose to:

- Start the simulation with the current settings;
- Check if the settings are correct;
- Automatically invoke the right methods on the indicated time steps. E.g. if the field ‘change call probabilities’ has the value 500, then at time step 500 the right method is called.

NetworkMonitor. This class has some methods to subtract statistics from the network and write them to a user-specified file. The most important method is called **registrate**, which tells what exactly to write to the file.

NetworkView. The class provides the view-window within the main window, in which the graphical representation of the network is displayed. Methods are there to display nodes, lines, highlighted routes, and mobile agents. These methods are called when **changed** messages are being sent from the model.

ScalesView. To draw the part of the main window which specifies the meaning of the node colours.

NetworkViewController. Is connected to the network view and specifies which method has to be invoked when the left mouse button is pressed - the right node interface has to be opened.

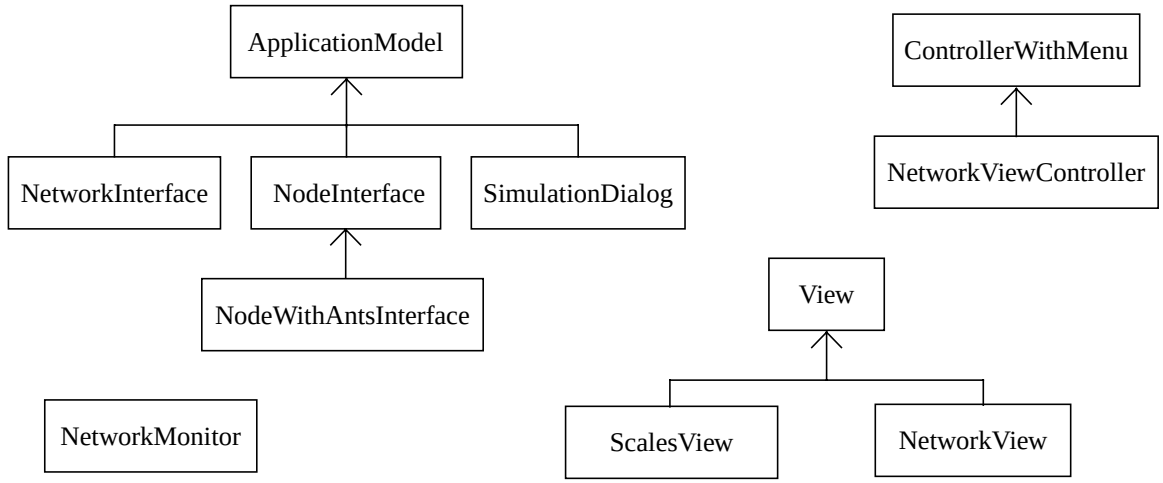
numbers in the array. For example, when the array is of size 4, and the exploration percentage is 20%, then 5% will be the minimum value every element has. The remaining values will be normalised to 80% in this particular case.

RandomNumber. **RandomNumber** generates natural random numbers according to probabilities indicated by an instance of **DistributionArray** or **PheromoneArray.** The numbers that will be generated are the indices of the array, and the values in the array represent the probabilities of their corresponding index being generated. At initialisation the array must be specified, but its values may change during the life-time of the **RandomNumber** object.

A.5 Network-Interface

The category Network-Interface contains the classes that define the user-interface (View and Controller in the MVC-model of VisualWorks).

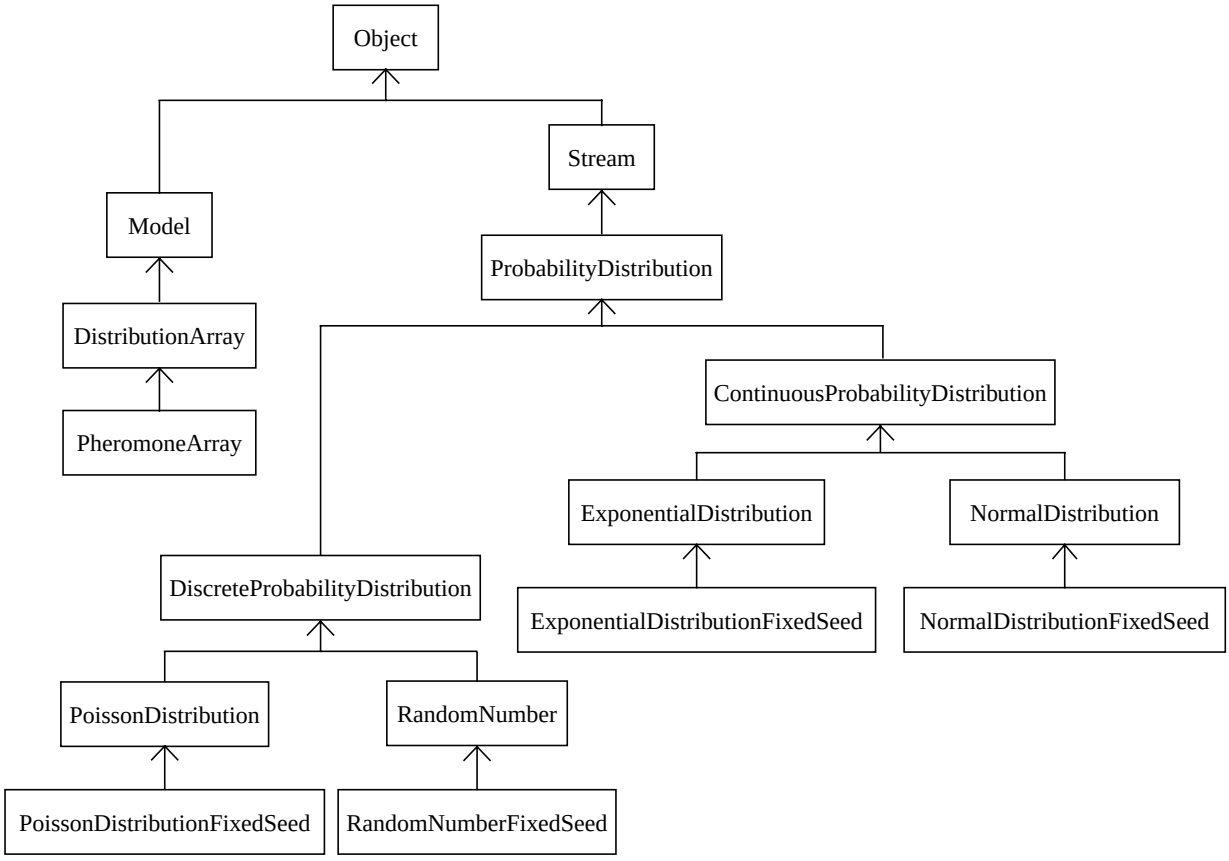
FIGURE 22. Class hierarchy of the user-interface objects. *ApplicationModel*, *ControllerWithMenu*, and *View* were predefined in VisualWorks.



NetworkInterface. Specifies the main window and menus of the simulation program, as it is shown in Figure 23 on page 73 (the window labelled ‘ABC Network Simulation Tool). The network interface automatically receives notification of changes in the model objects by means of the changed message, and the dependency mechanism [Lewis 95]. The widgets in the user-interfaces are then updated by means of methods that use so-called aspect adaptors. Further methods are written that has to be invoked when action buttons are pressed, or menu-items selected. These methods are able to control the simulation performed by the model, e.g. methods to:

- start and stop a simulation, or do one simulation time step;
- open a SimulationDialog window;
- start and stop monitoring simulation data to a file;
- reset the model;
- open a particular network topology;

FIGURE 21. Classes of category Probability-Distributions. Object, Stream and Model are predefined classes in VisualWorks. The three basic probability distribution classes were adapted from existing classes, obtained via Smalltalk libraries on the internet.



All of the above probability distribution classes provide initialisation methods, and a method called **next**, which generates the next random number according to the corresponding distribution. The classes ending on -FixedSeed provide initialisation methods for which the user can specify a seed for the random generator.

ExponentialDistribution. Supplies functions to generate real numbers according to the exponential distribution. At initialisation the μ parameter must be specified.

PoissonDistribution. Supplies functions to generate natural numbers according to the Poisson distribution. At initialisation the μ parameter must be specified.

NormalDistribution. Supplies functions to generate real numbers according to the Normal distribution. At initialisation the μ and σ parameters must be specified.

DistributionArray. Represents an array of real numbers whose sum is always equal to 1. These real numbers therefore represent probabilities. When one of these numbers is increased or decreased, all the numbers will automatically be normalised to sum to 1 afterwards.

PheromoneArray. Same as **DistributionArray**, now the feature of having a noise, or exploration percentage is added. This percentage is always equally divided over the

LoadAgentNew2. Other criterion for shortest route: Minimum sum of squared node utilisations.

AntAgent. Implements ants. The following instance variables are present:

- the id of the ant's source node, to know "what kind of pheromone to lay".
- the id of the ant's destination node, to know where to go
- its age in number of simulation time steps
- a random generator for randomly choosing neighbours according to the current node's pheromones, and whether this generator is seeded with a fixed number, or with the clock's milliseconds.
- the delay on the current node
- a unique identifier for each ant, depending on time of launch, node of launch, and number of launches on that time and that node before.

The methods of importance in the ant algorithm are:

- **determineWaitingTime**, a formula to calculate the delay;
- **gotoNextNode**, to go to a random next node, according to the current node's pheromone table for the ant's destination node;
- **performOperation**, to perform one complete operation an ant can do in one time step: If the delay is bigger than zero it does nothing, and else the ant goes to a next node and updates the pheromone tables (lays pheromones).
- **pheromoneIncrease**, a formula to calculate the amount of pheromone to be laid by the ant.

The mechanism of pheromone updates is implemented by the class PheromoneArray, which can be found in the category Probability-Distributions.

HandicappedAntAgent. This Ant does not lay pheromone and performs its route immediately instead of step by step. It is used for counting the number of steps an ant needs to go from one node to another, which will serve as a sample from a random variable.

A.4 Probability-Distributions

The category Probability-Distributions contains classes to generate and to help generate random numbers. The class **PheromoneArray** is also part of this category.

ability to move to desired neighbours, or to move to a neighbour in the direction of another ‘goal’ node, according to the routing tables. Agents are implemented as finite state machines - they have a possible number of ‘modes’, and each time step it performs the operation belonging to this mode[1]. This operation consists of a move to a next node, and eventually an operation on it.

ParentAgent. Parent agents launch load agents according to gathered information. The two possible modes an agent has are **gatherInformation** and **lookForOverload**. If in the mode **gatherInformation**, then the agent obtains the necessary data from its current node, and update its tables. If enough data is gathered, and if the utilisation of the node is higher than the utilisation history, then the mode becomes **lookForOverload**. In this mode the agent will be on its way to the node with the highest destination rate (according to its tables), and launch a load agent when there.

ParentAgentBackwards. This class is similar to ParentAgent, however the load agents that are launched by agents of this class, are updating routes in the other direction (as in the original proposal by Appleby and Steward).

ParentAgentDataField. Basically the objects of this class are records to contain data about nodes, kept in the parent agent’s tables. To compare destination rate histories, the methods **<** and **<=** are defined.

ParentAgentDataFieldBackwards. With overridden **<** and **<=** operators.

LoadAgent. A load agent updates the routes from all nodes in the network to its starting node, and self-terminates afterwards. It finds the paths to the start node that have the maximum smallest spare capacity on the path. It holds a table of permanent labelled nodes and a table of temporary labelled nodes, both containing objects of the class **LoadAgentDataField**. The variable mode can possess the values **addTemporaryNodes**, **onWayToPromoteNextNode**, and **ready**. In the mode **onWayToPromoteNextNode**, the agent is heading for the node of which the identifier is contained by the variable **nodeToBePromoted**, to promote this node. In the mode ready, the agent will go to its source node to unset the flag loadAgentWorking there. The mode addTemporaryNodes requires the sub-modes **onPermanentNode** and **onTemporaryNode**, contained in the variable **subMode**. During these sub-modes the agent travels between its last promoted node and its neighbours that the load agent does not yet have labels for, to label them temporary.

LoadAgentBackwards. This load agent updates the routing tables in the opposite direction of **LoadAgent**. Instead of updating the routes from all nodes in the network to its starting node, it updates the from its starting node to all nodes in the network, and the nodes in between.

LoadAgentNew. Load agent with other criterion for ‘shortest route’: It finds the route with the maximum value for minimum spare capacity divided by route length. Because updating routing tables can now only be done when the node is promoted, several methods had to be overridden, and two new methods had to be added.

1. The ‘perform-mechanism’ of Smalltalk / VisualWorks is used here. An instance variable mode contains the name of the method to perform. One step is the processing of a method with the call **self perform: mode.**

call generator is increased, and the method **nextCalls** is called to generate new calls for the network;

- a method to run the simulation in the background. In this way the interface widgets will still work.

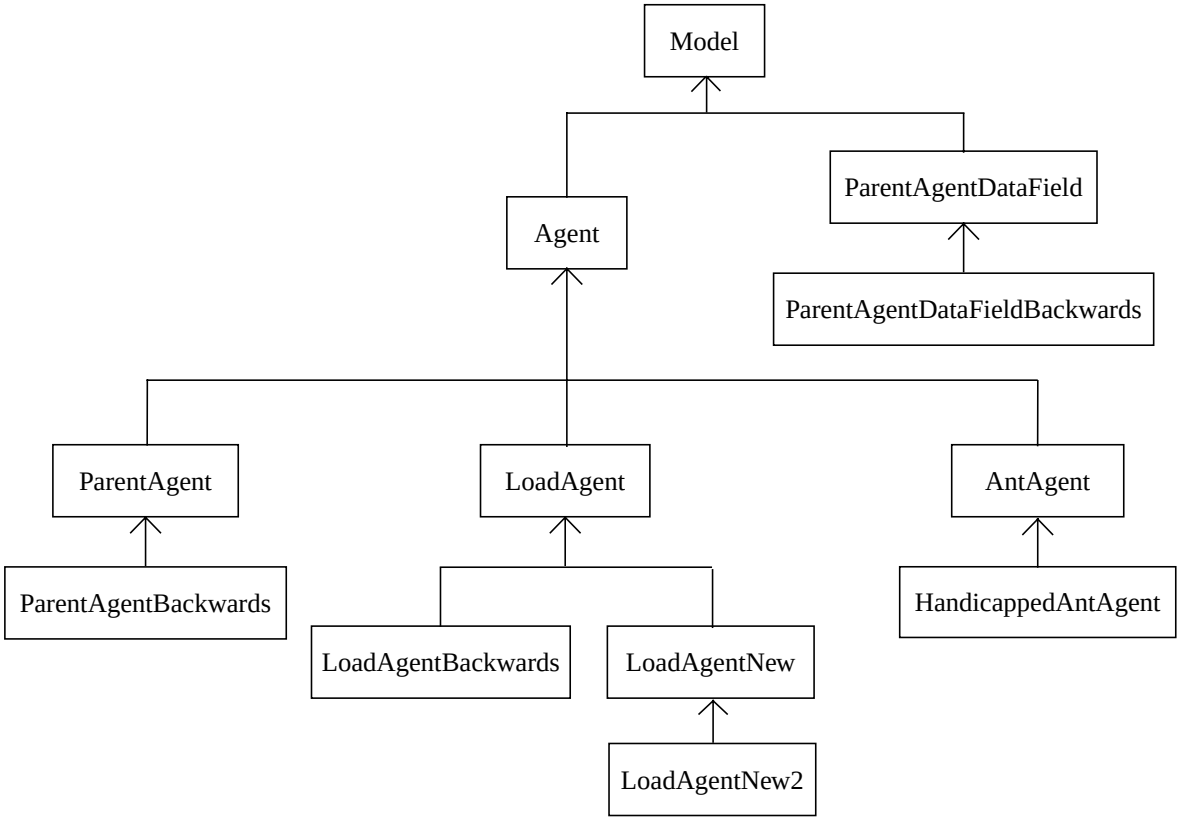
NetworkWithAnts. Some properties had to be added to the class Network to make it suitable to populate ants. For example a necessary initialisation period must be supported. **doTimeStep** and some initialisation methods had to be overridden.

RouteFinderPartialSolution. This class implements a backtracking algorithm to find a shortest route between two nodes.

A.3 Network-Agents

The category Network-Agents consist of classes to generate the mobile agents and the ants. Mobile agents are implemented by the **ParentAgent** and the **LoadAgent** classes. Ants are implemented by the class **AntAgent**. The class hierarchy is shown in Figure 20.

FIGURE 20. Classes of category Network-Agents. *Model* is predefined in VisualWorks. Arrows stand for inheritance



More information of these classes:

Agent. The class **Agent** supplies the basic property of mobility for every mobile agent and ant. It has access to its current node, and has methods implementing the

NodeWithAnts. This inherits the properties of class node, but replaces the routing table by tables of probabilities, called pheromone arrays. For every possible destination there is one pheromone-array of the size of the number of neighbours. The pheromone array for a certain node consist of probabilities. To go to a certain destination, an entry of the pheromone array indicates the probability to go to its corresponding neighbour node. Further a method called **launchAntAgent** is added.

GeneratedCall. An instance of **GeneratedCall** represents a call that is generated by a call generator or by hand, and is not yet placed on the network. It contains a source node identifier, destination node identifier, duration, start time, and accessing methods for these variables.

CallRecord. The objects of this class are the result of placing a generate call on the network. Additional data required here is the route of the call. A method is written to see if the call is expired and needs to be removed from the network.

CallGenerator. A CallGenerator object generates random calls on the network and keeps track of time. Its instance variables contain data like

- the average call duration;
- the average number of calls per time step;
- the network where it is generating calls on;
- a call distribution array, which contains for every node of the network, the probability of being the source or destination node of a call;
- stream variables to generate random number;s
- the time counter.

The most important method in this class is **nextCalls**. which generates a random number of calls (this number is Poisson distributed), all of which have a random duration (exponential distributed), and random source and destination nodes (distributed according to the call distribution array).

Another important method is **randomizeCallProbabilities** which generates randomly new call probabilities for each node, and normalises them to sum to 1.

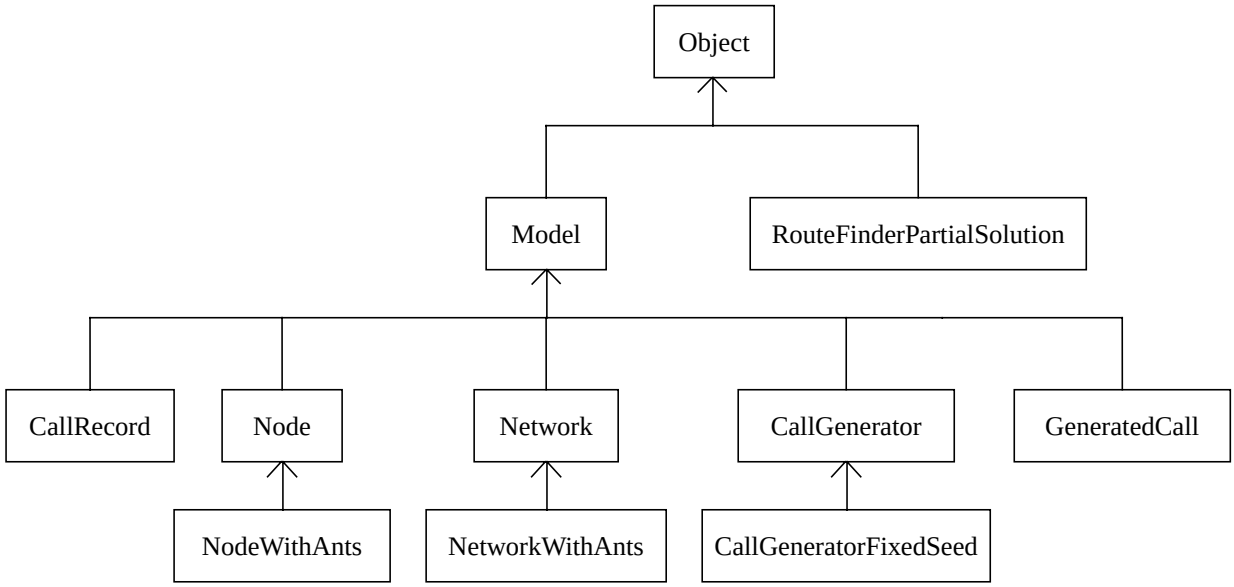
Network. This is the class that implements the simulated switch-based network. Its instances have access to its set of nodes, the present call records, and the call generator. Further it keeps track of the number of call failures and successes. It contains 47 methods. Next to a large number of methods that try to make life easier for other objects, it contains methods to:

- calculate the average route length;
- determine the route between two nodes;
- realise a generated call by making a call record of it, and placing it on the correct route of the network if enough spare capacity available;
- remove a call from the network;
- obtain statistics like the average node utilisation, and the standard deviation of the node utilisation;
- do one time step in the network. This method, called **doTimeStep** is called in a loop by the simulation interface. Old calls are removed, the time counter in the

A.2 Network-Model

The category Network-Model contains the classes that implement the basic simulation model, independent of the interface. It consists of the classes given in Figure 19:

FIGURE 19. Classes of category Network-Model. *Object* and *Model* are predefined classes in VisualWorks 2.5. Arrows indicate class inheritance. (i.e. all classes inherit from class Object)



More information about the classes of this category is given below.

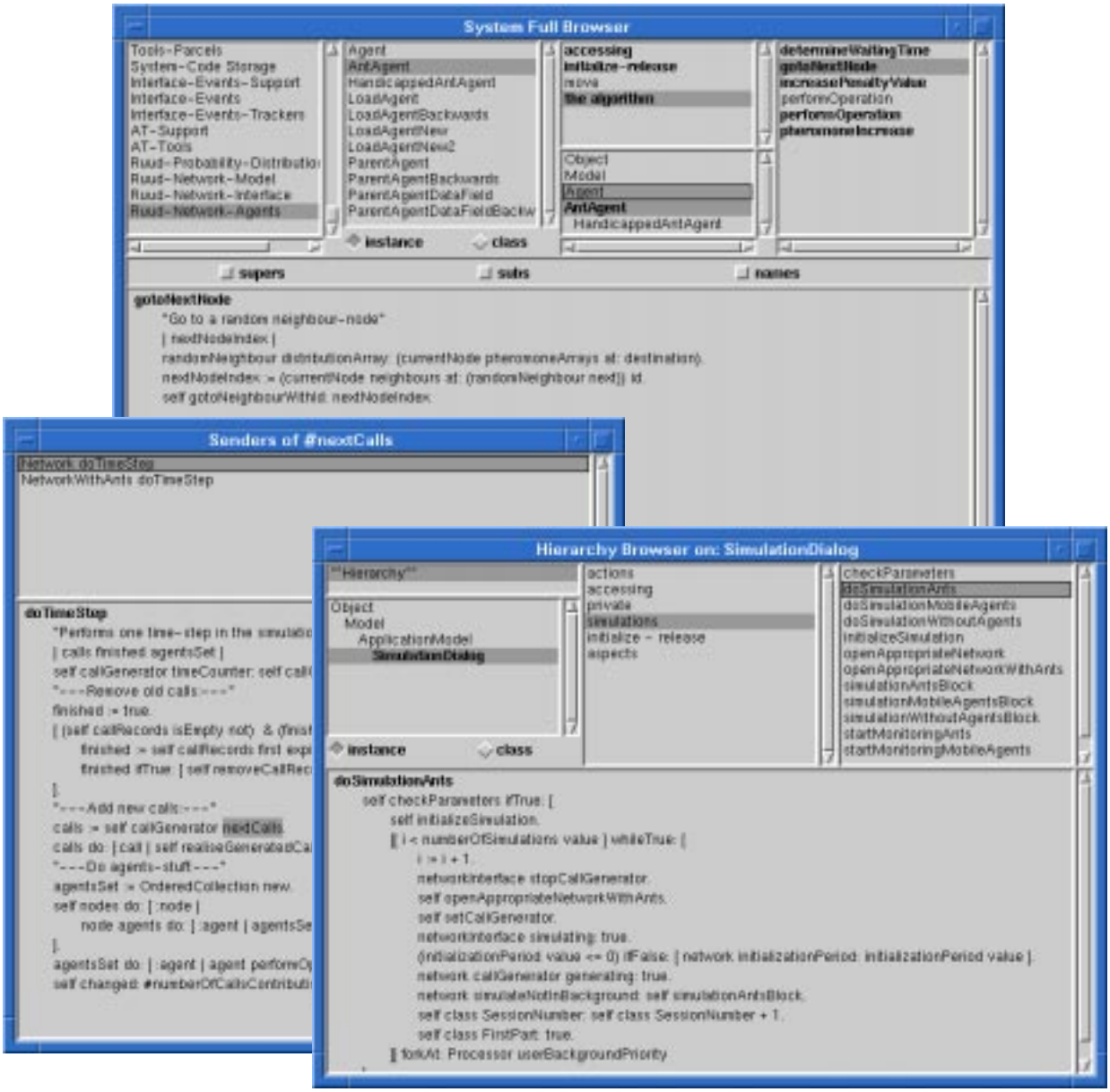
Node. A node represents a switching centre in the network. The node contains the following data (among others):

- Its identifier
- The set of nodes that are neighbours of it
- Its capacity
- Its available spare capacity
- The call records that go over the node
- A routing table: If an entry with index *x* in the table is equal to *y*, this means that to go to node no. *x*, you have to go via neighbour node *y*.
- Whether there is a load agent working for it
- The agents that are present on the node

The class Node provides 49 methods, sorted into protocols. For example, methods are written to:

- Obtain the node's sourcing rate and destination rate;
- Launch Parent- and Load agents;
- Adding and removing call records.

FIGURE 18. Three available browsers of Smalltalk code



The organisation of the code given in this appendix is according to the distinction in categories I made. There are four categories:

- Network-Model
- Network-Agents
- Network-Interface
- Probability-Distributions

Appendix A Implementational Issues

A.1 Introduction

The simulation has been written in the language Smalltalk-80, with development environment VisualWorks version 2.5 from ParcPlace systems. VisualWorks is a fully object-oriented programming environment, with a large scala of class hierarchies and tools to facilitate the design and implementation of user interfaces. It provides a framework called model-view-controller (MVC) to make a clear separation between the model and user interface of a program and to ease visualisation of the model.

Models in the simulation program are for example the network, the call-generator, each node and each call. Certain aspects of these objects notify the user interface when they are changed. After such a notification the user interface accordingly updates its views and widgets on the model. In this way it is possible to interactively view the network while calls are made and routes are changed by agents or ants. The interface allows the user to manipulate the model, by for example starting the call generator or modifying parameters of the experiment.

As Smalltalk is a fully object-oriented language, the generally accepted terminology from the field of object-oriented programming is used as much as possible.

Smalltalk code is hierarchically organised in categories, classes, protocols and methods.

- Categories are sets of related classes;
- Classes in Smalltalk are seen as special objects that provide methods to generate and specify other objects;
- A protocol is a set of related methods in a class;
- A method can be seen as a procedure on an object, or message sent to that objects.

Further one makes a distinction between class methods (for example the method **new**) and instance methods (messages that can be sent to objects), and between class variables and instance variables.

In this appendix I decided not to print any code I wrote for this research project, because of a number of reasons. The VisualWorks environment, together with Smalltalk libraries on the internet, provide a set of browsers that make it easy to go through the different levels in the hierarchy of code. Three of those browsers are shown in Figure 18.

As code in a browser is far more readable and searchable than a complete program listing, Smalltalk programmers rarely print listings. Methods and class definitions are generally short, and they can be found or edited by means of the system browser, hierarchy browser, or one of the other tools. A listing will be too long, and further the Smalltalk environment allows extensive reuse of code, and therefore any program listing is not complete. When you desire to browse the code of the simulation program, one can request it from the author. This appendix shows some brief descriptions of the different classes.

[Wolfram 84] Wolfram, Stephen. Universality and Complexity in Cellular Automata. *Physica 10D (1984), 1-35*.

A qualitative and quantitative distinction between four universality classes in cellular automata - fixed, periodic, complex and chaotic.

[Zaera 95] Zaera Tost, Nahum. Simulated Schooling: Collective Behaviours in Synthetic Fish. *Unpublished Master Thesis, School of Cognitive and Computing Sciences, University of Sussex. 1995.*

A research project performed at Hewlett-Packard Laboratories Bristol on the simulation of fish with the goal to get schooling-behaviour. Fishes are controlled by a neural network that is genetically evolved. Interesting results are obtained.

[Russel 95] Russel, Stuart and Norvig, Peter. Artificial Intelligence, a modern approach. Prentice Hall 1995. p. 31-35.

The book presents the state of the art in artificial intelligence, based on the concept of intelligent agents. Used also at Delft University for introductory course in AI.

[Schwartz 89] Schwartz, Mischa and Stern, Thomas E. Routing Protocols. *Computer Network Architectures and Protocols. 2nd Edition. Sunshine, Carl A. (ed.). Plenum Press New York / London 1989.*

Overview of current different routing strategies. Mainly on packet-switched networks.

[Sims 94] Sims, K. Evolving 3D Morphology and Behavior by Competition. *Artificial Life IV, proc. Of the Fourth International Workshop on the Synthesis and Simulation of Living Systems. Brooks, R.A. and Maes, P. (ed.). MIT Press 1994. p.28-39.*

A description of a system for the evolution of virtual creatures competing in a graphically represented three dimensional simulated world. Very nice mpeg-movies of these creatures can be found on the world wide web via the links ftp://think.com/users/karl/Welcome.html or http://www-uk.hpl.hp.com/people/jlb/Images/creatures-demo.mpg.

[Steels 94] Steels, Luc. Emergent functionality in robotic agents through on-line evolution. *Artificial Life IV, proc. Of the Fourth International Workshop on the Synthesis and Simulation of Living Systems, Brooks, R.A. and Maes, P. (ed.). MIT Press 1994. p. 8-16.*

An attempt to implement evolutionary techniques on a robot by means of competing processes.

[Stickland 92] Stickland, T.R., Tofts, C.M.N. and Franks, N.R. A path choice algorithm for ants. *Naturwissenschaften 79, 567-572. 1992.*

This article provides an example of a path choice algorithm by ants. The model is deliberately kept simple to investigate various conditions. Ants do not lay trails, but influence a bias (probability) in favour of the path they take when they return. Several results are put into graphs to show differences in parameter settings.

[Sutton 91] Sutton, Richard S. Reinforcement Learning Architectures for Animats. *From Animals to Animats. Proceedings of the first international conference on simulation of adaptive behavior. p. 288 - 296. The MIT Press 1991.*

Mentions the blocking and shortcut problem that also occurs in ants.

{Tanenbaum 88] Tanenbaum, Andrew S. Computernetwerken. *(Dutch translation of 'Computer Networks'). Prentice Hall 1988, Hoofdstuk 5 De Netwerklaag.*

Introductory book in computer networks, guided by the OSI model. The network layer implements routing, and therefore the corresponding chapter was consulted for this thesis.

[Wilson 75] Wilson, Edward O. Sociobiology, the new synthesis. Belknap/Harvard 1975. p.186.

Bible of the field of sociobiology. Indicated page goes into the principles of sema-tectonic stigmergy.

[Wilson 91] Wilson, Stewart W. The Animal Path to AI. *From Animals to Animats. Proceedings of the first international conference on simulation of adaptive behavior. p. 15-21 The MIT Press 1991.*

Wilson proposes a systematic approach to do research on artificial creatures, called animats, interacting with a possibly changing environment.

Certain cellular automata show that a dynamics of information has gained control over the dynamics of energy. It is argued that this is the case near a critical phase transition, from periodic behaviour to chaotic behaviour. Further it is suggested that this is what happens in living systems.

[Lewis 95] Lewis, Simon. The Art and Science of Smalltalk. Prentice Hall / Hewlett-Packard 1995.

A good introduction to the object-oriented programming language Smalltalk, and its underlying model-view-controller mechanism.

[Lumer 94] Lumer, Erik D. and Faieta, Baldo. Diversity and Adaptation in Populations of Clustering Ants. *From Animals to Animats 3, Proceedings of the third international conference of adaptive behavior. Cliff, D., Husbands, P., Meyer, J.-A., Wilson, S.W. (ed.). MIT Press 1994. p.501-508.*

Sorting behaviour of ant-like processes.

[Maes 93] Maes, Pattie. Behavior-Based Artificial Intelligence. *From Animals to Animats 2, Proceedings of the Second International Conference of Adaptive Behavior. Meyer, J.-A., Roitblat, H.L. and Wilson, S.W. (ed.). MIT Press 1993.*

Definition of behaviour-based AI as a novel approach to the study of intelligence. It compares this approach with traditional knowledge-based AI.

[Nolfi 94] Nolfi, Stefano; Floreano, Dario; Miglino, Orazio and Mondada, Francesco. How to evolve autonomous robots: different approaches in evolutionary robotics. *Artificial Life IV, proc. Of the Fourth International Workshop on the Synthesis and Simulation of Living Systems, Brooks, R.A. and Maes, P. (ed.). MIT Press 1994. p.190-197.*

Several examples to use evolutionary techniques in a simulation to evolve real world robot controllers.

[Ray 90] Ray, Thomas, S. An Approach to the Synthesis of Life. *Artificial Life II, Proceedings of the workshop on Artificial Life, Santa Fe, New Mexico, Langton, C.G., Taylor, C, Farmer, J.D., Rasmussen, S. (ed.). Addison Wesley 1990.*

The original presentation of Tierra, a virtual environment in which creatures of assembly code compete to survive by reproduction and mutation. Although not explicitly programmed, creatures become more advanced and notions like parasitism and immunity to parasites appeared in the simulation.

[Resnick 94] Resnick, Mitchell. Turtles, Termites, and Traffic Jams. Explorations in Massively Parallel Microworlds. *MIT Press, Cambridge, Massachusetts. 1994*.

A well written introduction to decentralised systems, technical background is not needed.

[Reynolds 87] Reynolds, C.W. 1987. Flocks, Herds and Schools: A Distributed Behavioral Model. *Computer Graphics Vol. 21, Part 4.*

A famous simulation of flocking birds (boids). The boids themselves are coded with hand crafted rules.

[Robson 73] Robson, Colin. Experiment, design and statistics in psychology. Penguin Books 1973.

Statistical tests to interpret experiments. Used for a description of Student's t-test.

[Hogg 91] Hogg, Tad and Huberman, Bernardo A. 1991. Controlling Chaos in Distributed Systems. *IEEE Trans. on Systems, Man and Cybernetics 21, 1325-1332*

First part is a short introduction on distributed AI. What problems arise? Example application: Two agents collaborating or competing to share two resources. Agents that perform best are rewarded according to some policy, for example by means of a genetic algorithm. It is shown that in this way an (almost-) optimal allocation can be obtained, instead of periodic or chaotic oscillations.

[Hölldobler 94] Hölldobler, B. & Wilson, E.O. 1994. Journey to the Ants. A story of scientific exploration. *Belknap Harvard 1994.*

I used to think that ants are dirty animals. After reading this book, you will find out that ants are the most interesting animals on earth. Many examples of different ants species and their behaviours.

[Horgan 95] Horgan, John. From Complexity to Perplexity. *Scientific American, Vol. 272, No. 6, June 1995.*

Destroying criticism of the fields artificial life and complexity. These fields do not present facts, lack clear definitions, and are overloaded with hype. No nuance, very biased.

[Johnson 94] Johnson, D. Brown, G.N., Botham, C.P., Beggs, S.L., Hawker, I. 1994. Distributed Restoration Strategies in Telecommunications Networks. *British Telecom Technology Journal, Vol. 12, no.2, April 1994.*

About a tool to evaluate several distributed restoration algorithms (DRAs) on networks on the British SDH network. The principle of distributed restoration is based on the fact that one of two nodes between a failing line sends an alarm, so that its neighbour nodes will search for new connections. These algorithms perform better than central protection algorithms and link protection algorithms (for each link a restoration plan).

[Kawata 94] Kawata, Masakado and Toquenaga, Yukihiko. From artificial individuals to global patterns. *TREE 9(11): 417-421. 1994.*

A brief introduction to Artificial Life.

[Koza 93] Koza, J.R. Genetic Programming. MIT Press 1993.

A well-known introductory book in genetic algorithms and genetic programming.

[Langton 87] Langton, Christopher G. Artificial Life. *Artificial Life, The Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems, Los Alamos, New Mexico, Langton, C.G., Taylor, C, Farmer, J.D., Rasmussen, S. (ed.). Addison Wesley 1987.*

The first large workshop on artificial life needs an introduction to the field, by the founder of it.

[Langton 90a] Langton, Christopher G. Preface. *Artificial Life II, Proceedings of the workshop on Artificial Life, Santa Fe, New Mexico, Langton, C.G., Taylor, C, Farmer, J.D., Rasmussen, S. (ed.). Addison Wesley 1990.*

Gives a definition of Artificial Life.

[Langton 90b] Langton, Christopher G. Life at the edge of chaos. *Artificial Life II, Proceedings of the workshop on Artificial Life, Santa Fe, New Mexico, Langton, C.G., Taylor, C, Farmer, J.D., Rasmussen, S. (ed.). Addison Wesley 1990.*

[Dijkstra 59] Dijkstra, E.W. A Note on Two Problems in Connexion with Graphs *In Numerische Mathematik vol. 1. 1959.*

The original article of Dijkstra's shortest path algorithm.

[Etzioni 93] Etzioni, Oren. Intelligence without Robots: A Reply to Brooks. *AI Magazine, winter 1993*.

A reply to [Brooks 91]. Etzioni argues that real world interaction is not necessary to obtain intelligent agents. Softbots, functioning in a complex and dynamic software environment are suitable candidates as well.

[Franks 89] Franks, N.R. Army Ants: A Collective Intelligence. *American Scientist, Volume 77, March-April. 1989.*

Extremely large numbers of army ants exhibit remarkable problem solving behaviour and form organisational structures, although one army ant is a very unsophisticated insect. Many examples are given: Regulating nest temperature, carrying items with more ants, massive emigration of a colony, efficient raiding of areas, laying short trails to food sources.

[Franks 94] Franks, Nigel R. and Tofts, Chris. Foraging for work: how tasks allocate workers. *In: Animal Behavior, 1994, 48, 470-472*

The observation that older ants do other work, farther away from their nest, than the younger ones, can be explained as an emergent property of the structure of the system.

[Gibbens 93] Gibbens, Richard J., Kelly, Frank P., and Turner, Stephen R.E. Dynamic Routing in Multiparented Networks. *IEEE/ACM Transactions on Networking, Vol. 1, No. 2, April 1993.*

The dynamic alternative routing based on dual parented networks.

[Grassé 59] Grassé, P.P. La reconstruction du nid et les coordinations interindividuelles chez *bellicositermes natalensis* et *cubitermes sp*. La théorie de la stigmergie: essai d'interprétation du comportement des termites constructeurs. *Insectes sociaux, tome VI, no. 1, 1959*.

Extensive descriptions of observations of termites serve as an introduction to the a new term: 'stigmergy'.

[Grimaldi 89] Grimaldi, Ralph P. Discrete and Combinatorial Mathematics. *Addison Wesley 1989, p. 523-530.*

Dijkstra's shortest path algorithm explained with good illustrations.

[Gutowitz 95] Gutowitz, Howard. Artificial-Life Simulators and Their Applications. *from internet-link: 'http://alife.santafe.edu/topics/simulators/dret/' Technical Report for French government 1995.*

An overview of the field of artificial life. The connection between artificial life and traditional computer science is discussed, as well as neural networks, evolutionary algorithms and cellular automata. A number of packages is analysed and large lists of research groups and world wide web resources are given.

[Hecht-Nielsen 91] Hecht-Nielsen, Robert. Neurocomputing. *Addison Wesley 1991*.

An introductory book on neural networks. Used at Delft University for their course on the subject.

[Brooks 91] Brooks, Rodney, A. Intelligence Without Reason. *Prepared for Computers and Thought, IJCAI-91, MIT, April 1991.*

A strong criticism on traditional AI research. It is stated that the work on computer architectures had an influence on our models of thought, that was too strong. The work on behaviour-based AI has produced new models of intelligence much closer to biological systems. A set of key ideas is presented, to do this style of AI research.

[Bruten 94] Bruten, Janet. Artificial Life and HP. *Internal Report Hewlett-Packard Laboratories Bristol 1994.*

Answer to the question: What could Artificial Life mean for a company like HP?

[Cliff 93] Cliff, Dave; Husbands, Philip and Harvey, Inman. Evolving Visually Guided Robots. *From Animals to Animats 2, Proceedings of the Second International Conference of Adaptive Behavior. Meyer, J.-A., Roitblat, H.L. and Wilson, S.W. (ed.). MIT Press 1993. p. 374-383.*

Results of a research project to use evolutionary techniques to evolve neural-network control architectures in a simulation of a visually guided robot, involving detailed models of the physics of an existing real robot.

[Collins 90] Collins, Robert J., Jefferson, David R. Antfarm: Towards Simulated Evolution. *Artificial Life II, Proceedings of the workshop on Artificial Life, Santa Fe, New Mexico, Langton, C.G., Taylor, C, Farmer, J.D., Rasmussen, S. (ed.). MIT Press 1990. p.579-601.*

The program AntFarm is an attempt to simulate the evolution of cooperative central-place foraging in ants. The simulated ants have a neural network trained and built with a genetic algorithm which operates on colonies of ants (one colony is seen as one super-organism). The fitness-function is positively influenced by food taken to the nest, and negatively by doing several actions. Very computationally expensive experiments (i.e. 8 days on a CM-5) lead to not very spectacular results.

[Dawkins 86] Dawkins, Richard. The Blind Watchmaker. Penguin Books, 1986.

Very good description of the updated version of Darwin's evolution theory. Using computer models as illustration, Dawkins shows that evolution is the only possible theory of how we got into existence.

[Deneubourg 89] Deneubourg, J.L. and Goss, S. Collective patterns and decision making. *Ethology, Ecology & Evolution 1: 295-311, 1989.*

This article explains a number of collective behaviours in animal groups. The understanding of this behaviour could be achieved by analysis of the interactions between individuals of the group. Relatively simple behaviours can lead to complex group behaviour. A widely observed social interaction is allelomimesis, which is roughly 'do what my neighbour is doing'. Allelomimesis is especially mediated by sight and chemicals. Behaviours similar to the behaviours of animal groups are obtained by mathematical modelling and computer simulation.

[Deneubourg 91] Deneubourg, J.L., Goss, S., Franks, N., Sendova-Franks, A., Detrain, C. and Chrétien, L. The dynamics of collective sorting robot-like ants and ant-like robots *From Animals to Animats. Meyer, J.A. and Wilson, S.W. (ed.). MIT Press 1991. p.356-365.*

Ants with simple probabilistic behaviours on the level of the individual, can sort objects on a grid collectively.

Bibliography

[Appleby 94] Appleby, S. and Steward, S. Mobile software agents for control in telecommunications Networks. *BT Technology Journal, Vol. 12, No.2*. 1994.

This article presents a novel approach for the control of distributed systems like telecommunications networks. It is based on mobile agents, computational processes which are capable of moving from node to node around a network. When using suitable programming guidelines, i.e. subsumption, an intrinsically robust and adaptable control system can be obtained.

[Bean 96] Bean, Nigel. Secrets of network success. *Physics World, February 1996.*

Treats several problems occurring in network routing, like instability. Further Braess' paradox is explained. This paradox shows that in some situations, even if a link (or street) is added to a network (or city), its call drop probability gets higher (or cars get more delayed).

[Beckers 92] Beckers, R., Deneubourg, J.L. and Goss, S. Trails and U-turns in the Selection of a Path by the Ant Lasius Niger. *Journal of theoretical Biology. 159, 1992. p.397-415.*

A series of laboratory experiments and accompanying computer simulations show how the ant *Lasius Niger* select the shorter of two paths between nest and food-source. Next to bi-directional trail laying, the mechanism of making U-turns on certain moments influences the path choice of other ants.

[Beckers 94] Beckers, R., Holland, O.E., and Deneubourg, J.L. From local actions to global tasks: Stigmergy and Collective Robotics. *Artificial Life IV, proc. Of the Fourth International Workshop on the Synthesis and Simulation of Living Systems, Brooks, R.A. and Maes, P. (ed.), MIT Press 1994. p. 181-189.*

This paper presents some experiments where mobile robots gather a number of objects and cluster them into a pile. Coordination of their movements is achieved through stigmergy.

[Belew 92] Belew, Richard K. McInerney, John and Schraudolph, Nicol N. Evolving Networks: Using the Genetic Algorithm with Connectionist Learning. *Artificial Life II, Proceedings of the workshop on Artificial Life, Santa Fe, New Mexico, Langton, C.G., Taylor, C, Farmer, J.D., Rasmussen, S. (ed.), Addison Wesley 1992. p.511-548.*

An extensive study of the possibility to combine genetic algorithms as an additional learning technique for neural networks.

[Bonabeau 94] Bonabeau, E. Theraulaz, G., Arpin, E. And Sardet, E. The Building behavior of lattice swarms. *Artificial Life IV, proc. Of the Fourth International Workshop on the Synthesis and Simulation of Living Systems. Brooks, R.A. and Maes, P. (Ed.), MIT Press 1994. p. 307-312.*

A presentation of building algorithms for artificial swarms, inspired by how wasp colonies build their nests. The insects move randomly on 2D or 3D lattices, using the local shapes they encounter to determine their individual building behaviours. Good example of the principle of stigmergy.

[Brooks 86] Brooks, Rodney A. A Robust Layered Control System For A Mobile Robot. *IEEE Journal of Robotics And Automation, Vol. RA-2, No.1, March 1986.*

This article is the original presentation of Brooks' Subsumption Architecture as an alternative approach to make control programs for mobile robots. Nowadays the principle of subsumption is widely applied in experimental projects.

Mathematical foundations. Although I think there is hope for a possible mathematical analysis of the behaviour of the artificial ants presented here, many phenomena in artificial life lack such an analysis explaining the systems behaviour.

Experience. A field like Artificial Intelligence is a settled discipline that has been there for decades. Artificial Life as a fields only has existed for ten years, and still has to prove itself to be applicable. Without my enthusiasm for artificial life I would probably not have been able to obtain the insights and new methods I presented in the previous chapters. Artificial life is a very exciting, but young field. There is a potential to solve problems in a different way, and it has been very interesting to take this approach. But the field of Artificial Life is not yet a real discipline, like many Artificial Intelligence subfields are. It will also not replace the other more traditional approaches, but provide a number of new approaches and increase our understanding of the world around us.

6.3 Thoughts

Despite the fact that there are little theoretical foundations of artificial life, one should perhaps note the following: For systems residing in increasingly complex environments like world wide interconnected telecommunications systems with many interacting components, there will maybe not always be much value on theoretical proofs of correctness. Maybe people should be more open for the possibilities provided by alternative distributed approaches that exhibit implicitly programmed emergent behaviours. The engineering of these systems will probably be more based on experience, feeling, and natural examples, or on evolution over a longer period of time.

[Franks 89] argues that intelligence is an emergent property of collective communication. Ants communicate through pheromones. Neurons communicate thousands of times faster than natural ants, due to the differences in speed of the chemical messengers. As opposed to natural ants, the artificial network ants walk ‘with the speed of light’, and the system as a whole therefore ‘learns’ much faster. The speed of the artificial ants presented here should be in the same order as the speed of artificial neural networks. The ABC system is very suitable for parallel implementation too. Although natural ants communicate slowly, their methods of survival proved to be as effective on an evolutionary time scale[1], as the collective of neurons in the brains of mammals. Summarising these thoughts, one could say that there might be a lot of benefit to gain by applying ant-based techniques in computer environments.

I hope that this research project provides new insights for people working in the area of intelligent agents or distributed AI. This project has demonstrated an example where agents do not need to be ‘intelligent’ themselves to collectively exhibit seemingly intelligent behaviour. Extending ant-like algorithms to situations like telecoms networks which are not found in nature will also increase our understanding of the abstract and general abilities of such algorithms over and above those applications found in nature. I hope that this increase of knowledge will in turn assist and inform biologists studying social insects.

1. The total biomass of ants in the world, is approximately the same as the total mass of humans. Their population is in the order of ten thousand trillion [Hölldobler 94].

Failures in network. The pheromone tables do not only represent very good routes, but also contain information about the relative merits of alternative possibilities if something goes wrong. My simulations have so far been confined to examining the use of this information in dealing with node congestion; sudden node (or link) failure and restoration also needs to be simulated to examine the abilities of the ants to deal with these contingencies. The ability to cope with the insertion of new nodes and links during network extension is also a topic of interest.

Other application areas. I believe that ABC systems can be used to solve a large variety of optimisation problems, e.g.:

- distributing loads of interconnected processors on parallel machines and managing inter-processor communication for complex programs;
- material flow in production environments;
- optimal routing on integrated circuit-boards;
- organising public transport schemes.

Braess' Paradox. Another interesting research topic might be whether the ABC-system is able to beat Braess' Paradox [Bean 96]. This well-known paradox tells us that when adding a road in a city, traffic actually is likely to become slower in some networks of roads. In communication networks there might be more calls dropped when a link is added. Routes are considered to go to user-equilibrium - that is the situation where blocking (or call failure) probabilities are the same on all routes used. This situation arises for example in car traffic when every car tries to optimise its own route. Ants however, are eusocial (truly social) and therefore somehow different from cars. The ants in the ABC-system will lay a stronger trail if they find a fast route, and in this way encourage their fellow ants to take it too. (People in) cars tend to be more thinking about themselves. As network traffic tends to take the same route as ants, Braess' paradox might get beaten because of this different approach.

Mathematical analysis. The underlying principles of the behaviours of the ant-based control system are not well understood. Parameters had to be chosen on the basis of experience instead of mathematical analysis. Whether it is possible to formulate a mathematical model that allows us to do predictions about the systems' behaviour, remains to be seen.

6.2 The downside

Throughout this thesis I have been advocating for programming by means of emergent behaviour as a possible approach to solving problems that could not be solved before. There have been criticisms to these approaches as well [Horgan 95], and there are some downsides.

Predictability. The behaviour of programs that make use of emergent behaviour, is often only known when the system is finished. It is a lot easier to prove the functionality of 'normal' top-down designed systems in advance.

Guidance instead of control. It is difficult if not impossible to directly, centrally control a large number of interacting objects. Instead, e.g. 'swarms' need to be guided by triggers to which they respond, like herds of sheep are guided by a dog. There is the additional problem of upgrading systems that are highly distributed - how do you reach all parts of the system?

6.0 Conclusions

A completely decentralized adaptive control system for telecommunications networks was implemented which made use of emergent collective behaviour arising from the interactions between mobile objects modelled on ants, and tables on network nodes.

The ant-based control (ABC) system was compared with a mobile software agent approach. Load management agents were regularly launched in the system to update the routing tables according to a least cost criterion. This mobile agent system proved to be very reactive to changes in call patterns, and is robust as well, due to its distributed nature. The ABC system however performed better, and the reasons for this were pointed out.

The principles of the ant algorithm are simple and general. I believe that the general framework for ant-based solutions presented here is capable of solving load balancing problems in large networks, both circuit switched and packet switched. We do not yet know exactly how the statistical and topological properties of the network influence the ideal parameter settings. But as shown here, even tuning parameters by hand can lead to a well balanced system.

The balance obtained is a good example of emergent organisation. The individual ants are remarkably simple, but the resulting structures enable very efficient use of a resource like a telecommunications network. Three possible types of adaptation to the characteristics of such networks were identified, and ABC systems show themselves capable of good performance on all three types.

6.1 Future investigation

Unfortunately, nine months was too short to discover all relevant mechanisms and properties of ABC systems. I hope to provide people who decide to do further research on this topic with some directions:

Influence of parameter settings. Much investigation of the basic principles of the ant algorithm remains to be done. So far, my experiments have not yet enabled me to make statements that are sufficiently supported by statistics about the influence of the number of ants used in the simulations. I also do not know exactly how variations in the ants' influence on the pheromones affect the system, or about the effects of the size of the delays imposed on the ants. Most choices in this work are based on a relatively small number of experiments.

Scaling. It would also be useful to investigate the performance of the algorithm on extremely large or very small networks; the large networks will tell us about scaling, and the small networks might assist our understanding of the basic processes which make the algorithm work.

Other mechanisms. Another intriguing possibility is to use 'probabilistic routing' of calls. Here routes of calls, or perhaps a proportion of calls, would not be chosen according to the largest probabilities in the pheromone tables, but randomly according to these probabilities. A mechanism that is assumed not to be used by natural ants, but could be useful here, is laying 'anti-pheromone'. One could let ants directly decrease probabilities in the pheromone tables in particular circumstances, rather than increase them.

launched on a particular node, another one can not be launched until the first agent finishes its job, because otherwise they would interfere with each other.

Robustness. Malfunctioning in the system might cause a mobile process such as an ant or agent to crash. If an ant crashes, this will not have a significant effect on the performance of the algorithm at all. However, if a load agent or parent agent crashes, this affects the future launching of load agents. This therefore has to be detected, and special measures have to be taken to restore the damage.

Ants can be considered as simple mobile data elements being processed upon by the node, rather than mobile computational processes, and so no problems will occur due to corruptly programmed ants. As an individual ant has a small influence on the system, damage caused by a corrupt ant will be limited. However, a load agent on its own can change all the routes to its source node, and can therefore cause much more damage if its state is corrupted.

Computational issues. Ants are likely to require more computation on the nodes of the network than the mobile agents, due to the extensive use of random generators. Further, with ant-based control, nodes need to allocate more space for their pheromone tables than is needed when normal routing tables are used. However, these issues do not affect bandwidth or switching capacity, which is our main concern.

Bidirectional routes. During the simulations of the ant controlled system, the route from one node to another tends often to be the same as that in the opposite direction. This is probably due to my mechanism of trail laying, where ants from complementary source and destination nodes mutually reinforce one anothers' trails. The mobile agents do not have this property. At first sight, this property might seem to be disadvantageous for good load balancing, but I believe that this will only make a significant difference in small networks.

Circular routes. In principle, ABC systems have the potential to yield circular routes. However, this situation was not observed, except when the noise was extremely high, or the initialisation period much too short. The mobile agents as implemented here are guaranteed not to result in circular routes.

We note that the standard deviations are quite high for the situations when call patterns are changed and load balancing is stopped. I think this is due to the fact that the routing or pheromone tables at the moment of stopping agents or ants are partly adapted to the situation of that instant. This might be a temporary situation that is relatively exceptional, so that freezing the tables does not yield good static routing information. On the other hand it also might be a temporary situation more representative of an average situation, producing acceptable results when the tables are frozen.

In the ABC system, the balance between dynamic adaptation to temporary situations and finding good static routes could clearly be adjusted by manipulating the delay function and the pheromone update function. In my experiments I fixed these parameters on an empirical basis at levels which appeared to give low levels of call failures when adaptation was complete; I did not systematically explore their effects on the dynamics of adaptation.

I believe that some of the power of ABC comes from the fact that the system stores information not only about good current routes, but also about good current and recent *alternative* routes. The mobile agents do not have a representation for alternative possibilities, and base their routing decisions only on temporary situations, which limits their capability to adapt to a more general, average pattern of node utilisations. I also suspect that with the mobile agents a particular problem occurs which has been recognised in the field of network routing before [Kelly 1995]: Making a new routing decision based on a temporary situation might result in a longer route that is only beneficial for a short period. Calls over this route put a higher demand on network resources and may cause a number of subsequent calls to be lost. As the demand for node capacity increases, more traffic has to be rerouted. This leads to a kind of cascade effect. In this way short term benefits are outweighed by the longer term costs. The ABC system naturally seems to find a good balance between adaptation over short and long time periods.

5.5 Ants versus mobile software agents

As well as the performance advantages presented in the sections above, ants have a number of other qualitative and quantitative advantages over mobile agents, as well as some disadvantages.

Consuming network resources. An ant hardly requires any bandwidth on the network: It only holds its age, and its source and destination identifiers (together with the fact ‘I am an ant’). The mobile agents hold a number of relatively large tables and therefore require much more bandwidth than ants.

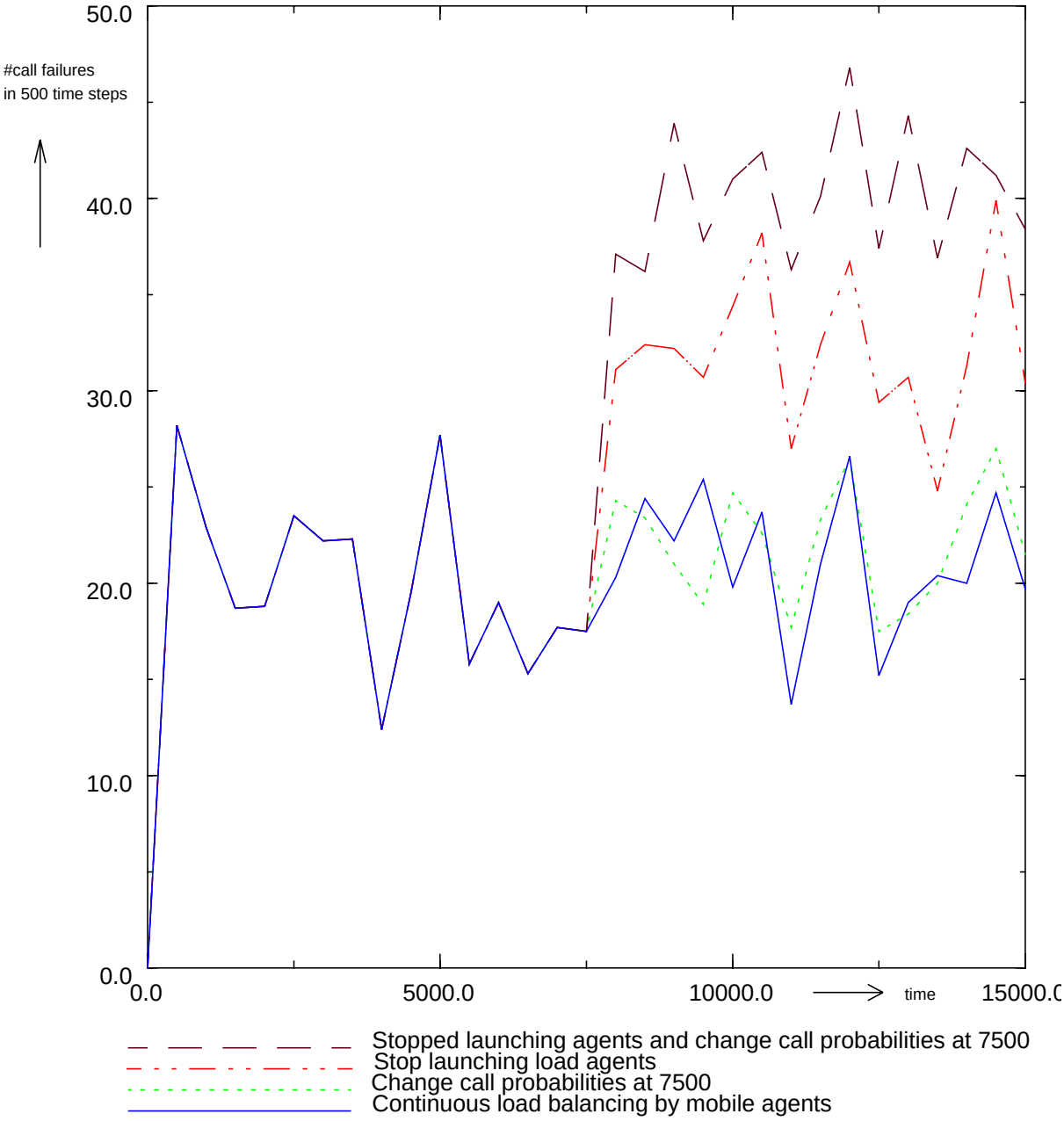
A property of the mobile agents from BT is that more load agents tend to get launched as the load on the network increases, to do re-routing. This might be just the moment when you do not want any more agents on the network, as it is already congested with calls. In contrast, congestion-dependent delay of the ants temporarily reduces ant traffic at congested locations. Having said that, fewer mobile agents are required than ants. The numbers of mobile agents used is of the order of tens, whereas ants are used in hundreds.

Limits to the number of agents. In principle, there is no limit to the number of ants that can be used in a network, because ants do not interfere with each other. The number of load agents on a network is much more limited. Once a load agent is

adapts to the new set of call probabilities. The reaction of the mobile agent system to changing call probabilities is much faster.

- In both types of control system, the graph of the situation when there is continuous load balancing with no change in call probabilities lies lower than the one where load balancing is stopped. This indicates that some advantage of the systems comes from continuous dynamic adaptation to temporary situations.

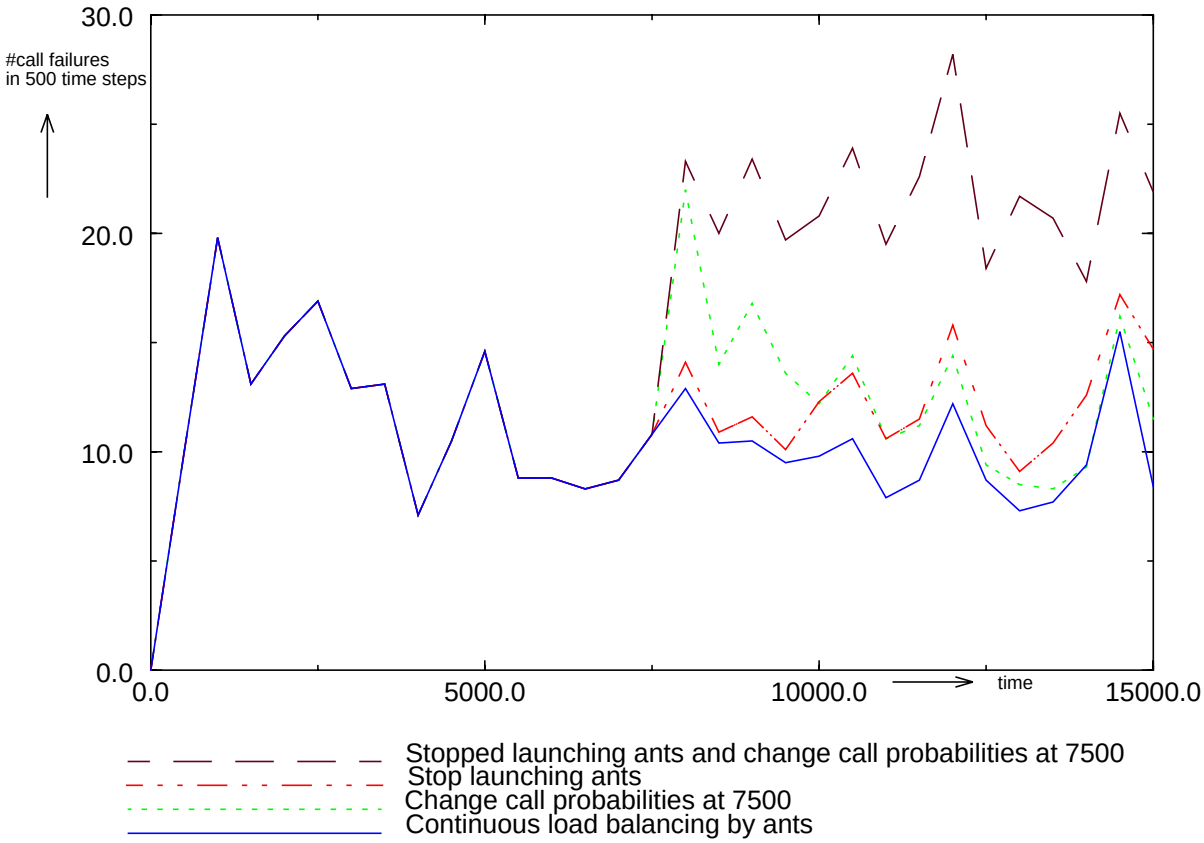
FIGURE 17. Performance of the improved mobile agents. The average number of call failures of ten experiments during every 500 time steps of the simulations are plotted



routes on the network to temporary situations as well. The same can be said about the agents, which seem in fact to be more sensitive to these temporary situations than the ants, as the difference in performance between the two agent experiments is relatively larger. Close observation of the network while running the simulation also confirmed my impression of useful reaction to temporary situations for both the ants and the agents.

Both type of systems thus appear capable of all three types of adaptation. The operation of the different components of adaptation for the ABC system may be seen in Figure 16, which shows the performance of ants with 5% noise (the graph for 0% noise looks similar and is shown in Figure 15). The corresponding graphs for the mobile agents are shown in Figure 17. All graphs are on the same scale. Note again that the peaks are the same because there is averaged over the same ten call blocks under each condition.

FIGURE 16. Performance of ants with 5% noise. The average number of call failures of ten experiments during every 500 time steps are plotted.



- By comparing the figures with the upper graph of Figure 15, one can see that even if the call probabilities are changed and load balancing stopped, in both types of control system there is on average a better set of routing tables than with the fixed shortest paths determined without load balancing technique. This illustrates the adaptation to the network topology.
- It is obvious that as soon as the call probabilities are changed, the ABC system starts producing an increased number of call failures, after which the system

5.4 Static solution or dynamic adaptation

One of the most interesting questions raised during the simulations on the 30-node network was whether the control systems were converging to a good static set of routing tables that was 'learned' from the statistics of the network, or whether they were constantly adapting to changing situations. A good static set of routing tables would combine information about the network topology and the call distribution statistics; routes would be sufficiently short, but would avoid the nodes likely to become congested with those particular call statistics. I think that three different forms of adaptation are possible:

- Adaptation to the network topology
- Adaptation to the call probabilities of the nodes
- Adaptation to temporary situations caused by the randomness of the call patterns

Adaptation to network topology. When I speak about adaptation to the network topology, I mean how well the control system adapts to the distribution of loads on the nodes that arises as a consequence of the specific topology. For the ABC system, the system converges to short paths during initialisation; for the mobile agents the system is initialised with the shortest paths. Due to the topology alone (and independent of a particular set of call probabilities) routing calls over these paths will already lead to congestion at certain nodes. The system will adapt to this congestion by changing its pheromone tables. Although adaptation to the distribution of network loads is a response to both the topology and the call probabilities, it is possible to get some insight into how well the system with an arbitrary set of call probabilities adapts to the topology alone. This insight can be obtained by inspecting the results of the experiments where dynamic load balancing is stopped at the same time as the call probabilities are changed. Any useful adaptation of the control system can then only be in relation to the topology, which is the only factor left unchanged.

For both the agents and the ants, the performance under these conditions is better than the experiments with fixed shortest path routing tables and no load balancing at all. Further one can clearly see that the ABC system performs better than the mobile agents (8.03% call failures for the improved agents versus 4.29% and 4.37% for the two ant experiments); this indicates that the ABC system adapts better to the load distribution caused by the topology alone.

Adaptation to call probabilities. To see how well a method performs when adapting to a combination of topology and call statistics, one can consider the results where the launching of agents or ants is suddenly stopped, but the call probabilities remain the same. Here the ABC system performs better than the mobile agents (6.43% call failures for the agents versus 2.11% and 2.48% for both ant systems). The results are also better than those discussed above, involving only adaptation to the network topology. The size of the difference gives an indication of the influence of the call probabilities.

Adaptation to temporary situations. The performance of ants and agents on adapting to temporary situations is indicated by the differences in performance under unchanging call probabilities of the conditions where load balancing is either continued or stopped. The situations where ants are launched after 7500 time steps perform better than those in which launching is stopped. Although the ABC system has adapted to call probabilities and to the topology, routes are still changed frequently in response to temporary situations. Because this results in improved performance, we can take this to indicate that ants are dynamically adapting the

FIGURE 15. Performance of four load balancing techniques, with unchanging call probabilities. The results are averages of ten experiments plotted every 500 time steps. Similar peaks are due to the fact that the same ten call patterns are used for every condition.

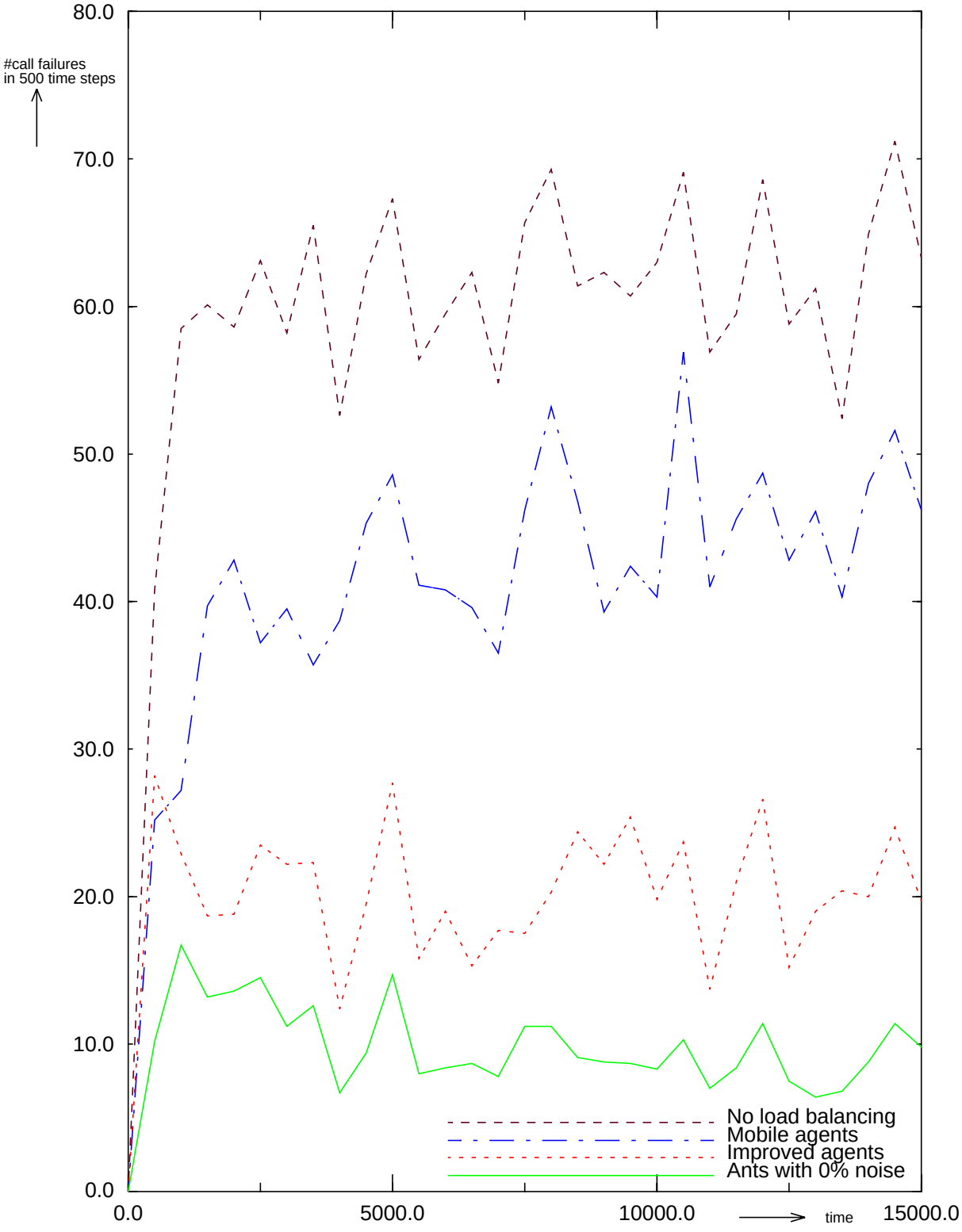


TABLE 12. The mean percentages and standard deviations of call failures after stopping load balancing and changing call probabilities

	Mean	Standard dev.
No improved mobile agents after 7500	8.03%	2.88%
No ants (0% noise) after 7500	4.29%	2.06%
No ants (5% noise) after 7500	4.37%	2.27%

The Student t-test (2-tailed) for related samples was performed on the call failure data (the complete numbers and a description of the Student-t test are given in Appendix C); it was possible to pair observations derived from the same call sequences in the test period. 1B was paired with 1B, 2B with 2B, and so on. The following differences were found at the 0.01 significance level:

- Under all conditions the improved mobile agents gave significantly better results than the original mobile agents.
- All ant experiments gave significant better results than the corresponding experiments with the improved mobile agents.
- In the case of unchanging call probabilities, ants without noise gave better results than ants with 5% noise.
- The experiments with the 0% noise ants gave significantly better results in the simulations without changing call probabilities than in the simulations with changing call probabilities.
- All experiments with ants and agents were significantly better than the experiments where there was no dynamic load balancing.
- When the parent agents stopped launching load agents when the monitoring of call failures started (time step 7500), the results of the simulations were that significantly more calls fail in the experiments with and without changing call probabilities.
- Stopping the ants causes more call failures than leaving the ants working.
- In the situations with unchanging call probabilities, stopping the ants produces better performance than leaving both kinds of mobile agents working.
- It was not possible to tell with sufficient significance whether noise is helping the ABC system to quickly adapt to the new sets of call probabilities.

To illustrate the performances of the different algorithms I also measured the number of call failures during every 500 time steps of each run for both the adaptation and test periods for the unchanging probability conditions. The averages over the ten runs under each condition are shown in the graph of Figure 15. Three of the graphs represent the situations where there is continuous load balancing, the fourth represents no load balancing. The graph for the ants with 5% noise is not depicted as it is very similar to that for the ants with 0% noise. Note that all four graphs have the same peaks. This can be explained by the fact that every condition is tested with the same ten call blocks.

TABLE 9. The mean percentages and standard deviations of call failures after stopping load balancing for unchanging call probabilities

	Mean	Standard dev.
No improved mobile agents after 7500	6.43%	2.17%
No ants (0% noise) after 7500	2.11%	0.60%
No ants (5% noise) after 7500	2.48%	0.69%

I was also interested in how well the load balancing systems will deal with a sudden change in call probabilities. I examined this by allowing the system to adapt to an adaptation block from one set of call probabilities, and then testing it with a test block from a different set of call probabilities.

The experiments as presented above were repeated, but now at time step 7500 the call probabilities and call patterns of run *x* changed to the call probabilities and call patterns of run *x*+1 in the former experiments; for example, after adaptation on block 1A, the system would be tested not on block 1B, but on block 2B. The experimental design is illustrated in Table 10.

TABLE 10. Experimental design 2

adaptation (0-7500)	test (7500-15000)
1A	2B
2A	3B
:	:
10A	1B

The mean percentages of call failures in these experiments were:

TABLE 11. The mean percentages (ten experiments each) and standard deviations of call failures for changed call probabilities:

	Mean	Standard dev.
Without load balancing (fixed, shortest routes)	12.53%	2.04%
Original mobile agents	9.24%	0.80%
Improved mobile agents	4.41%	0.85%
Ants (0% noise)	2.72%	1.24%
Ants (5% noise)	2.56%	1.05%

And the same experiments where agents and ants were no longer launched after time step 7500 (see Table 12):

TABLE 6. Generation of call sequences for adaptation and test. For example, 5A stands for the call sequence from time steps 0 to 7500 generated by the fifth set of call probabilities.

call probability set	adaptation (0-7500)	test (7500-15000)
1	1A	1B
2	2A	2B
:	:	:
10	10A	10B

- Each run consisted of an adaptation period (block A) and a test period (block B). During the adaptation period the load balancing system was allowed to adapt to the call statistics. I had found that any adaptation which takes place is substantially complete after 7500 time periods.
- During the test period (time steps 7500 to 15000) the network performance in terms of call failures was recorded. The experiments have been performed for: a fixed routing scheme without load balancing; the mobile agents; improved mobile agents as explained in Section 3.2.1; ants without noise; and ants with 5% noise. The experimental design is illustrated by Table 7 and the results in Table 8

TABLE 7. Experimental design 1

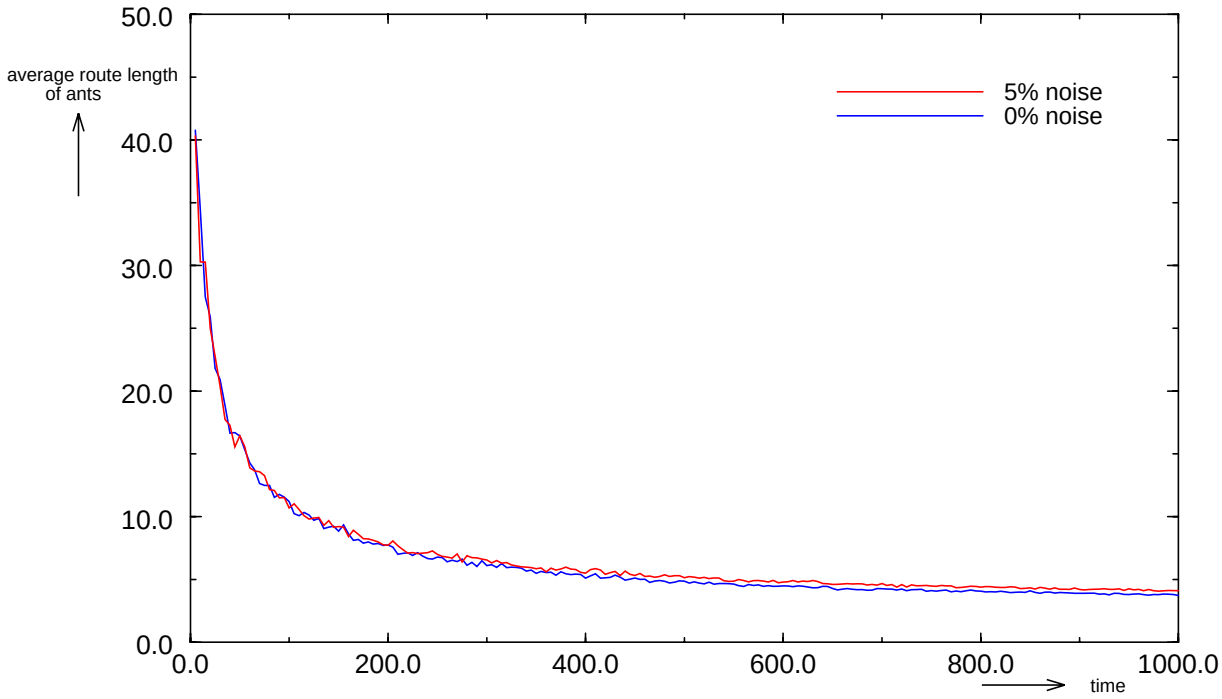
adaptation (0-7500)	test (7500-15000)
1A	1B
2A	2B
:	:
10A	10B

TABLE 8. The mean percentages (ten experiments each) and standard deviations of call failures for unchanging call probabilities

	Mean	Standard dev.
Without load balancing (fixed, shortest routes)	12.57%	2.16%
Original mobile agents	9.19%	0.78%
Improved mobile agents	4.22%	0.77%
Ants (0% noise)	1.79%	0.54%
Ants (5% noise)	1.99%	0.54%

In order to find out the extent to which the results are due to the dynamic actions of the load balancing system, rather than to the convergence to a good set of routing tables specific to a particular set of call probabilities, I repeated some of these experiments, but this time I froze the routing tables for each method after the adaptation period; i.e. I stopped launching mobile agents or ants. The results are shown in Table 9.

FIGURE 14. Energy reduction in the ABC system during initialisation period



5.3 Final experimental design and results

Both adaptive routing schemes are compared with a non-adaptive routing scheme with fixed routing tables that are algorithmically optimised to yield the shortest paths (in terms of the number of intermediate nodes). Arguments why I expect this routing scheme to perform less well than the adaptive approaches are given in Section 2.2.2.

The simulation presented me with some practical problems. Because of the initial random selection of call probabilities, the random generation of calls, and the random lengths of calls, the variability between runs was very high. In order to overcome this, many runs would have to be averaged; since each run was very time consuming, an adequate number of runs would take a prohibitively long time. It was therefore decided to use a repeated measurements experimental design, in which each condition is tested on the same dataset; because of the reduction in variability, pairwise comparisons between conditions can then yield good information from a relatively small total number of runs. It proceeded as follows:

- 10 sets of call probabilities were generated using the method of Section 5.1.2.
- Each set of call probabilities was used to generate a call sequence lasting 15000 time steps.
- Each call sequence was split into two blocks A + B, each of 7500 time steps.

The routing tables are initialised so that the length of every route, i.e. the number of nodes on the route, is minimal.

5.2 Convergence of the ABC system during initialisation

Before doing extensive performance tests of the ABC system, we first take a look whether the algorithm converges during initialisation. Will the routes that arise be sufficiently short after a certain number of time steps?

To test the convergence, a notion of energy in the ABC system is defined. In the first instance one would tend to think of the average route length (according to the highest probabilities) as a measure of energy, but in the beginning routes will often be circular, so that the route length is impossible to measure. Therefore we define energy as the average length an ant will walk from any node to any other node on the network.

How will this average length be measured? For one ant there are infinitely many routes possible, as there is no limit to the number of times an ant might visit a node. Therefore an approximation of the energy is measured by sending a special ant from every node of the network to every node of the network, counting the number of ‘hops’ it needs, and dividing it by the squared size of the network (= number of launched ants). These special ants will not have any influence on the pheromone tables, and complete there whole random walks in one time step. The obtained value is then an approximation for the average number of steps an ant needs to go from its source node to its destination node. Note that the theoretical minimum for this network is 3.07 (the route length is now measured in number of links).

An initialisation period of 1000 time steps was run to show how the energy developed. Every 5 time steps the average route length for ants was measured. The resulting graph is given in Figure 14. Note that the average route length according to the highest probabilities will always be lower when there are no circular routes anymore: The reason for this is that these special ants can also take ‘worse’ routes, due to their random walk.

selection from a suitable distribution at the start of every run, and lie between 0.01 and 0.07. After generation these probabilities are normalised to sum to 1.

- The capacity of each node is 40 calls, so that every call using a node increases the utilisation (or decreases the spare capacity) of that node by 2.5%.
- During every time step of the simulation, an average of 1 call (Poisson distribution) is generated with an average duration of 170 time steps (exponentially distributed). This means that the average number of calls on the network will be 170, once the traffic pattern has built up.
- The average length (total number of nodes) of the shortest route between two nodes is 4.07 (computed for this network with a specific procedure). With fixed shortest-path routing tables, each of the 170 calls will use 2.5% of the capacity of an average of 4.07 nodes. As there are 30 (number of nodes) times 40 = 1200 'capacity units', the average utilisation of the nodes will be 170 x 4.07 / 1200 = 57.7%.

5.1.3 Ants parameters

The parameters I chose for the ABC system are:

- The speed of the ants is one node per simulation time step (unless they are delayed on a particular node).
- I chose to let every node launch an ant with a random destination on every time step of the simulation.
- The probabilities are updated as explained in Section 3.2.1, and according to the following formula, where *age* stands for the number of time steps that passed since the launch of the ant:

$$\Delta p = \frac{0.08}{age} + 0.005$$

- The *delay* in time steps that is given to the ant is a function of the spare capacity *s* of the node:

$$delay = \lfloor 80 \cdot e^{-0.075 \cdot s} \rfloor$$

- The initialisation period, that is the period during which the ants initialise the routes on the network without traffic, is between 250 (no noise) and 500 (5% noise) time steps.

5.1.4 Mobile agent parameters

The parameters used to test the mobile agents were as follows;

- The speed of the agents is basically the same as the speed of the ants: every time step of the simulation an agent performs its task on its current node and moves to the next node.
- The parent agent takes the last 4 visits to each node into account to calculate the destination rate history.
- The global utilisation history is 60; the last 60 visits count in the calculation of the average utilisation.
- The destination ranking table has size 15. This means that if this table is smaller than 15, the parent agent will gather more information around the network
- The number of parent agents is 2.

5.0 Experiments

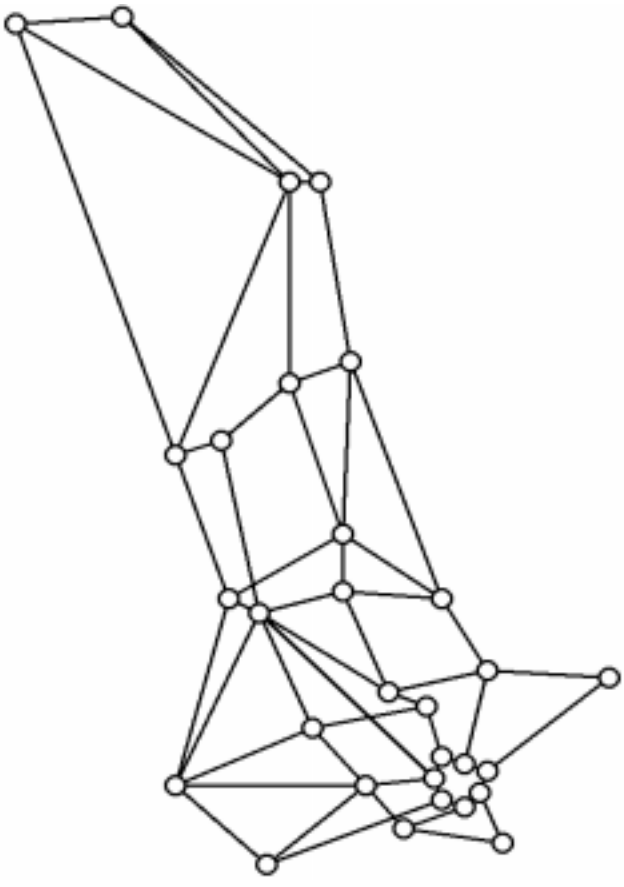
This chapter contains the parameter settings, design, results and conclusions of the experiments to test the algorithms of chapter 3.0 and chapter 4.0.

5.1 Parameter settings

5.1.1 Network topology

The 30-node network topology of Figure 13 was chosen because this is a realistic interconnection structure of a possible switch-based network. It is also the same topology as was used in [Appleby 94], and is in fact the structure of the British Synchronous Digital Hierarchy (SDH) network. The interconnection structure is an irregular mesh that is interesting because it makes traffic management a complex and difficult task.

FIGURE 13. This network topology is the same as the interconnection structure of the SDH network of British Telecom and provides a realistic network topology.



5.1.2 Simulation parameters

The parameters concerning the simulation and call statistics are:

- When a call is set up, each node in the model of Figure 13 has a certain probability of being an end point of the call. These probabilities are generated by

- The speed of the agents. The network simulation is a simulation of a distributed system on a single processor. Therefore one has to define how much work each part of the system can do in one simulation time step.
- The length of the parent agent's destination-rate history. How many past visits to the node are taken into account in this calculation? And how much do they count?
- The length of the parent agent's global utilisation history.
- The size of the destination rate ranking table of the parent agent.
- The number of parent agents in the network.

The routing tables are initialised so that the length of every route, i.e. the number of nodes on the route, is minimal.

4.5 Summary

In this chapter a version of a previously proposed decentralised network management method has been presented. Routing tables on the network are changed by mobile agents moving from node to node. The agents make their decisions based on local information and the information they gathered earlier.

The manner in which the agents were programmed was inspired by the subsumption architecture of Rodney Brooks. A layered control architecture was made, each layer implemented by a different species of agent. The lowest layer was implemented in the load management agent. Load agents organise routing tables to make better routes to the node on which they start their job, according to Dijkstra's algorithm. The guidelines to program according to Brooks' subsumption required thorough debugging of the load agents, before making a higher layer; parent agents. These agents launch load agents according to heuristic rules and information gathered around the network. This way of programming facilitates the making of complex programs in a straight forward way.

- *A destination rate ranking table*. This table contains a ranking of nodes according to their destination rate history.

When the agent encounters a node with a higher utilisation value than the agent's utilisation history, it assumes that traffic management is needed. The ranking table tells the agent which node is most used as a destination node of calls. The traffic to this node is likely to be the cause of some network overload. The agent then goes to this node and launches a load agent, unless the node has already got a load agent working for it. After reaching this node and launching a load agent, the parent agent continues its cycle of gathering information, detecting where new traffic management is needed. Note further that the parent agent looks at the destination rates, whereas the original mobile agents looked at the sourcing rate of nodes. This is due to the difference of direction in which my load agents update the routing tables.

4.2.1 Alternative tasks for parent agent

When testing the mobile agents on a network simulation, certain aspects of real networks are abstracted. A typical task that parent agents should perform in a real network is to detect where load agents have crashed, for example with help of the load agent's lifetime, kept on its starting node. Where the lifetime of the load agent exceeds the average lifetimes of other load agents with a certain safety margin, the parent agent might assume that the load agent could not return to its starting node because it has crashed. The parent agent therefore launches a new load agent. The possibility of crashing of an agent is not modelled in this simulation. Therefore the mechanisms to correct this are not tested in this study.

4.3 Higher levels of control

To manage the population of parent agents, a third level of control could be added. This level of control is provided by 'parent monitors', static processes that run on the nodes. The task of a parent monitor is to keep the population of parent agents on a desired level. Each parent agent registers itself on the parent monitors of a node. According to heuristic rules and the data these parent agents provides, a parent monitor might assume that more parent agents are needed. In the same way as parent agents decide whether a load agent is crashed, a parent monitor assumes if a parent agent is crashed. This means that a parent monitor checks every now and then when the last visit of a particular parent agent occurred. According to time intervals between those occurrences and a safety margin, a parent monitor decides to launch a new parent agent. As parent monitors might become corrupted and launch too many parent agents, parent agents should be able to terminate themselves if they notice that there are to many of them.

In the network simulation presented here, parent monitors are not implemented. Because parent agents are launched by the user and do not crash in the model, parent monitors are not required. Nevertheless the concept of parent monitors as a third level of control offers a good illustration of the advantages of this pseudo-subsumption architecture. This way of programming allows us to make control systems for complex and changing environments in an incremental way. The result is a robust control system with seemingly complex (intelligent) behaviour.

4.4 Parameters

The space of possible parameter settings is large; arbitrary choices will have to be made according to experience with previous experiments:

will not only update the routing table from the starting node to the newly promoted node, but also the tables of all nodes on the route to this node. This possible advantage will not be present in the approach presented here. However, the load agents will be faster in my approach and, as stated before, will not generate invalid routes.

4.1.5 Another criterion for 'shortest route'

An improvement to the load agent was made by storing the total sum of squared utilisations of all nodes on the route from that node to the agent's source node, instead of the largest spare capacity of the route. (The node utilisation is the percentage of the node's capacity that is occupied by calls.) The load agent then promotes temporary nodes with the smallest sum of squared node utilisations in its records. In this case the routing tables can only be updated when the node is being promoted, because when a node is still temporary it may be possible to find a better solution for that node later. The agent finishes as soon as all nodes have been promoted.

The reason why I chose a different criterion for 'shortest route' was that I observed relatively long routes in simulations where load agents maximise the minimum spare capacity. A call on such a long route occupies more nodes and this additional demand on network resources may cause subsequent calls to fail; other load agents may then amend the route to follow an even longer path. My improved algorithm counteracts this by taking the spare capacities of all nodes in the route into account, and leads to shorter routes (routes with fewer nodes). Note further that by squaring the utilisation, the relative influence of heavily utilised nodes is increased.

Note that now the routing tables of a node can only be updated when this node is being promoted. Therefore the statements made in Section 4.1.3 are not valid for this type of load agent.

4.2 The Parent Agent

The second level of control in the network is provided by parent agents. They travel around the network and launch load agents where network management is needed. The decision to launch a load agent is made on the basis of information gathered, and a set of heuristic rules.

Therefore the parent agent travels randomly around the network to gather information about the different nodes. At each node the agent visits, it records the following data fields:

- *The traffic destination rat*e. This is the number of calls that have the node as their destination
- *The utilisation of the node*, which is the percentage of the node's capacity that is used.
- *The destination-rate history*, which is the average destination rate of the last *d* visits to the node.

Further it records the following global information, when stepping around the network:

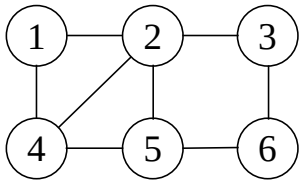
- *The utilisation history*, which is the average of node utilisations of the last *m* visited nodes.
- *The number of nodes it has visited so far.*

The above situation might seem unrealistic, as load agents will tend to find approximately the same routes. However testing this method on a network with 30 nodes (for example the network of Figure 13), with the same kind of simulations as will be presented in Section 5.1.1, confirmed that this problem indeed arises. Circular routes occurred after a few thousand time steps.

There is one way to overcome this problem, and that is to let the load agent set a flag on the destination node to which it is going to update the route. When another agent visits this node, it is not going to update the routes to it. After having done this, the agent should return to this node to undo this flag.

There is a second reason why updating the routes in the way described in [Appleby 94] may lead to undesired side-effects, in this case caused by only one load agent. This is also best explained with an example.

FIGURE 12. An example network



Consider the network in Figure 12. Some of the current routes are as follows:

- The route from 4 to 6 is 4-5-6.
- The route from 1 to 6 is 1-4-5-6.

Imagine a load agent launched at node 4. A new route may be found from node 6 to node 4, as node 5 might be congested. Consider the situation in which the agent will travel back to update the route to node 6. The new routes will be:

- The route from 4 to 6 is 4-2-3-6.
- The route from 1 to 6 is 1-4-2-3-6.

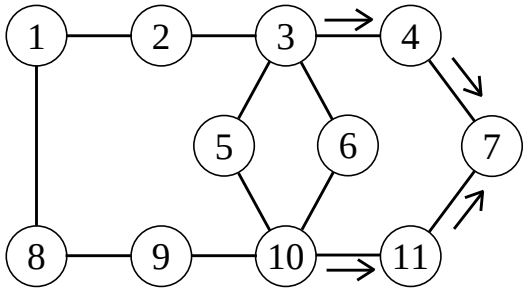
The side effect of this update is the change of the route from node 1 to node 6: The part ‘1-4-2’ could be directly replaced by ‘1-2’. The use of node 4 is then a waste of switching capacity. This is only a tiny example. Experiments with a much larger network with 30 nodes and many load agents resulted in routes that go from one side of the network to the other and back, until finally the destination node was reached.

To a limited extent, the same situation will arise temporarily if a load agent is launched on node 6 on the network in the example. It might visit node 2 earlier than node 1, and for a short period of time, the route from node 1 to node 6 will be the same as above. In this case however, it will be corrected as soon as the load agent reaches node 1. In the other situation, where the routes are updated in the other direction, the route will stay that way until a new load agent is launched on node 1. In this case the agents often will have had much more time to make many more strange routes, using nodes unnecessarily.

The reason why Appleby and Steward chose to update the network as they did was to let load agents have a beneficial effect elsewhere in the network: A load agent

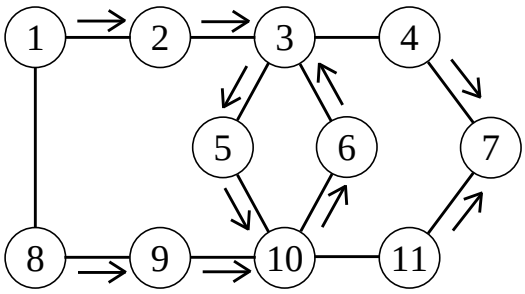
agents launched on different nodes may find different routes between these nodes during the overlap of their life times. While they are updating the routing tables, invalid routes may occur. This is due to the fact that two load agents might be updating routes to the same node at the same time.

FIGURE 10. An example network



Consider for example a (part of a) network like the one in Figure 10, and suppose there are two load agents working on it, according to the approach in [Appleby 94]. Load agent 1 is launched on node 1 and load agent 8 is launched on node 8. Both agents have recently promoted node 7 and are moving back to their starting node, to update the routes to node 7 on the nodes they encounter when moving back to their starting node. As the agents are running on a dynamic network, the spare capacities on the nodes are continuously differing. Therefore it might well be possible that the routes that both agents found are different. So suppose that load agent 8 is on node 3. The route to node 7 has just been updated, and is indicated by the arrows. At approximately the same time, load agent 1 is on node 10. This situation might occur for example when agent 1 would have found node 4 with a relatively low spare capacity, and agent 8 might have found node 11 congested. The next step for agent 8 is to go to node 6 to update the route to 8, and for agent 1 to go to node 5 to do the same. There are no problems so far. Now agent 8 moves to node 10 to update the new route to 7 (via node 6) and agent 1 steps to node 3 to make the route to node 7 go via node 5. The problem has arisen now. The final situation is given in Figure 11: A circle has occurred. If there is a call generated from nodes 1, 2, 3, 5, 6, 8, 9 or 10 to node 7, it will never be realised. The routes that the agents meant to be updating were: 1-2-3-5-10-11-7 and 8-9-10-6-3-4-7.

FIGURE 11. The new situation:



All nodes are visited now. All routingtables are updated, so the agent goes back to node 1 to inform it that it has finished, and terminates. The new routes to node 1 are:

2-1
3-2-1
4-3-2-1
5-4-3-2-1
6-2-1
7-1
8-7-1.

As node 6 is the node with the least spare capacity available, the agent automatically took care of avoiding node 6 while updating the new routes. The route from 5 to 1 is the clearest example.

4.1.3 Directly updating routing tables when labelling temporary nodes

The approach described above is based on, but not identical to, that described in [Appleby 94]. One of the differences is described in this subsection; another one in the next.

In the BT work, routing tables are updated when the node is labelled permanent. When a new node is being promoted, and new temporary nodes are added, the agent will check to see if the route via the current node has a higher spare capacity than that indicated in the record. If this is the case then the record will be altered to indicate the new, higher capacity and show the newly promoted node as the contributing neighbour. This situation will never occur. When a node is already temporary, the spare capacity that is indicated by the label will always be greater than or equal to the spare capacity of the route via the other node. In the original algorithm of Dijkstra, this is not the case, as Dijkstra's original algorithm deals with minimising the *sum* of the weights of the edges between nodes, instead of finding the route where the minimum capacity is maximal.

Therefore it is possible to do the route update immediately when making the node temporary, instead of waiting until the node is made permanent as a better route will never be found. In this way the route updates will be made at a much earlier stage than in the approach of BT. The only disadvantage is that the load agent needs to know the size of the network to be able to know when it has finished its job. However, the network size can be found in the routing tables of the nodes anyway. Note that this way of updating the routes therefore only accounts for this specific minimum cost criterion.

4.1.4 Updating the routing tables in the other direction

The major difference with the BT work is the direction in which the routing tables are updated. In Appleby and Steward's approach, after promoting a new node, the load agent travels back to its starting node, updating the route to the newly promoted node for every node on this route. In the approach presented here, only the route to the start node needs to be updated. There are two reasons for choosing this alternative.

The main problem occurring when updating the routing tables in the other direction comes forward when running the algorithm in a dynamic simulation. This means concurrently running several load agents on a network with continuously changing call patterns, implying continuously changing spare capacities. In this case load

TABLE 3. Records of load agent in step 3

node#	Temporary Nodes	Permanent Nodes	Route to start node
1		∞	-
2		90	1
3	40		2
4			
5			
6	25		2
7		85	1
8	30		7

Now the agent makes node 3 permanent. Node 4 will become temporary, with spare capacity min(40,80) = 40. See Table 4.

TABLE 4. Records of load agent in step 4

node#	Temporary Nodes	Permanent Nodes	Route to start node
1		∞	-
2		90	1
3		40	2
4	40		3
5			
6	25		2
7		85	1
8	30		7

The final step is to promote node 4, see Table 5.

TABLE 5. Records of load agent after final updates

node#	Temporary Nodes	Permanent Nodes	Route to start node
1		∞	-
2		90	1
3		40	2
4		40	3
5	35		4
6	25		2
7		85	1
8	30		7

TABLE 1. Records of load agent in step 1.

node#	Temporary Nodes	Permanent Nodes	Route to start node
1		∞	-
2	90		1
3			
4			
5			
6			
7	85		1
8			

Now the agent goes to the temporary node with the largest spare capacity in the table. This is node 2. This node is made permanent, and the neighbours of this node that have not yet been visited are made temporary. Node 3 will have the label 40, because that will be the smallest spare capacity on the route to 1. In the same way node 6 will be labelled 25. For both nodes, the route to the start node, node 1, will be via node 2. The situation is shown in Table 2.

TABLE 2. Records of load agent in step 2.

node#	Temporary Nodes	Permanent Nodes	Route to start node
1		∞	-
2		90	1
3	40		2
4			
5			
6	25		2
7	85		1
8			

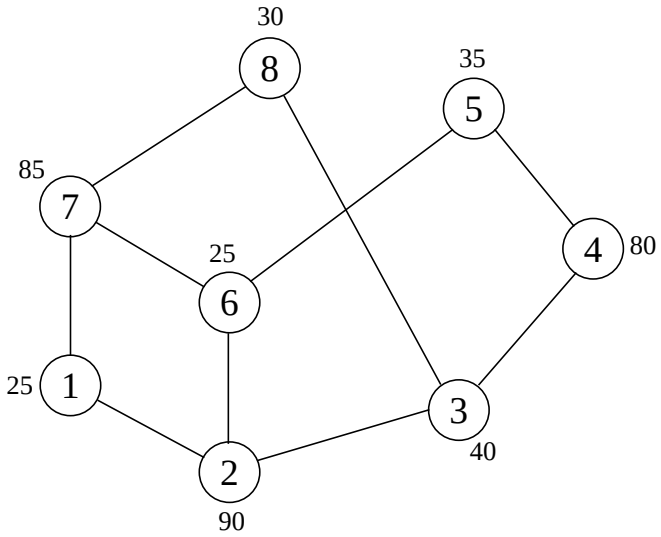
The temporary node with the largest spare capacity is node 7, and will become permanent. Node 8 is the only neighbour of node 7 that is not yet visited by the agent, and therefore becomes a temporary node. The spare capacity is min(85,30) = 30. The agent will update the route from node 8 to node 1, which will go via node 7. The new records are given in Table 3. The load agent is at node 8.

in the route into account. (When two temporary nodes are labelled with the same spare capacity available, the load agent makes an arbitrary choice which of these nodes to promote.)

4.1.2 The load management agent illustrated

To illustrate the algorithm, a possible network configuration is given in Figure 9. The numbers in the nodes represent the node numbers. The numbers outside the nodes represent their spare capacities. Each node has routing tables with an entry for each possible destination in the network, which represents the neighbour node on the route to this destination.

FIGURE 9. An example of a network (to explain the algorithm of the load management agent).



Suppose a load agent is launched on node 1. The node will set its flag ‘load agent working’ to true. The agent will initialize its records by putting the number ∞ in the permanent record for node 1. Having done this, the agent visits node 2 and node 7, to add temporary records for them, and to update the routingtables on the route to node 1. The agents’ records in this stage are given in Table 1. The third column contains the update of the routing tables that has been made. The agent is on one of the temporary nodes.

minimum spare capacity on the route is used in the original algorithm; and the sum of squared node utilisations for my version of the load agent.

4.1.1 The algorithm of the load agent

The load agent using the minimum spare capacity on the route as a shortest path criterion works as follows:

Travelling over the network the load agent makes a distinction between two kinds of nodes: permanent and temporary labelled nodes. The agent maintains lists with records of both kinds of nodes, with the following data-fields:

- The identifier of the node
- The largest spare capacity of the route from that node to the agent's source node. This is the route where the minimum spare capacity is maximal.
- the neighbour of the node on this route.

The goal of the load agent is to update the routes from every other node in the network to the agent's source node. Having done that, the agent is finished and terminates. The algorithm is as follows:

1. When a load agent is launched on a node, the node sets a flag ‘load agent working’. The agent starts by creating a permanent label for its source node, with spare capacity infinity and no contributor to the best route. The agent does not have records yet for any other nodes. Hereafter it visits all neighbour nodes, to create temporary records for each neighbour. In this first step the spare capacities in these records are equal to the spare capacities of the nodes themselves. The entries in the routing tables of the nodes will then be updated. This part of the initial step is also trivial: the next node in the route to the source node is the source node itself.
2. The next step is to promote one node from temporary to permanent. This will be the temporary node with the largest spare capacity in the agent's records. The agent goes to this node and moves the record for it to the list of permanent nodes. The agent then visits all of this node’s neighbour nodes that it does not have records for, and creates temporary records for them. For each of these records the contributing neighbour is the newly promoted node, and the spare capacity is the least of the neighbour nodes' own spare capacity and that recorded for the newly promoted node. Here the routing tables are updated again: the next node in the route to the source node is the new permanent node. At this stage the list of temporary records now consists of temporary records from this step, and from the earlier steps.
3. Again the temporary node with the largest spare capacity on its route to the source node is visited and made permanent. Its neighbours become temporary, their routing tables are updated, so that the agent is ready to promote another node. Steps 2 and 3 are repeated until the agent has visited all nodes in the network. At this point all routing tables have been updated, so the agent goes back to its starting node to undo the flag ‘load agent working’, and dies in peace.

The new routes that are found by a load agent are the optimal routes given the information gathered by the agents: the route from every node on the network to the agents’ starting node has a maximised minimum spare capacity. However, traffic patterns change while the agent is working, and so the information gathered by the agent need not accurately reflect the situation at a given moment - i.e. the routes are not guaranteed to be optimal. Note further that the agent only looks at the minimum spare capacities on the routes, and does not take the spare capacities of other nodes

4.0 Mobile Software Agents

Mobile agents might provide another way to solve load balancing problems in a distributed manner. In 1994 two researchers from British Telecom (BT), Stephen Appleby and Simon Steward, described a method to organise routes in telecommunications networks with mobile agents [Appleby 94].

To obtain a robust system, the agents were constructed in a similar way to autonomous robots based on the Subsumption Architecture (see Section 1.5.1). However, instead of implementing the different levels of task achieving behaviour as processes running on the agent, they proposed an approach in which they implement these levels by means of different species of agents. Every species will provide a level of competence and recognise and respond to a different set of patterns in the network. The different species are not able to *subsume* each others behaviours, and therefore it is actually not completely valid to call this architecture subsumption as Appleby and Steward did, it is more a good hierarchical way of programming.

In this mobile agents approach, there are three ‘species’ of agents: Load management agents, parent agents and parent monitors. The lowest level of control will be provided by the load management agent, in an algorithmic way. It will search for better routes from nodes in the network to the node where it is launched. Parent agents provide the second level of control. According to heuristics and information gathered on the network, a parent agent can decide that network management is needed and therefore launches load agents on these places. A higher level will consist of processes residing on the nodes, that monitor the parent agent population, again according to heuristics.

To test the mobile agents method, it has been implemented in the network model described in Section 2.0. On a number of points the method of Appleby and Steward had to be adapted, especially concerning the lowest level of control; the load management agent. Differences between the approach presented here and their approach, as well as the reasons for this, are described in Section 4.1.3 to Section 4.1.5.

4.1 The Load management agent

The load agent provides the lowest level of control in the network. As in the programming of mobile robots with subsumption, the lowest level has to be implemented first and debugged thoroughly.

A load agent is launched on a particular node and optimises the routes from all other nodes to that source node. It does this by visiting every node in the network, recording the current spare capacity, and amending the routing tables. Two methods were investigated, each with a different kind of load agent. In the first, load agents were made to find the route with a minimum bottleneck i.e. they maximise the minimum spare capacity on the route, as in the BT work. In the second a version was implemented in which they minimise the sum of squared node utilisations. The reason for this will become clear later. Both methods will avoid nodes with the lower spare capacities when choosing the new routes

The algorithm that is used to find the new routes is a version of Dijkstra's shortest path algorithm [Dijkstra 59] or [Grimaldi 89]. As a criterion for ‘shortest path’ the

- The initialisation period, that is the period during which the ants initialise the routes on the network without traffic.

3.3 Summary

A load management technique for networks has been presented, that is modelled on the trail laying and following abilities of ants. Routing tables in telecommunications networks could be replaced by tables of probabilities. These probabilities also indicate good routes, and have more information about how good alternatives are. To obtain a good set of probabilities, mobile objects move around on the system to update them. These objects are modelled on ants, and the way they update the probabilities is similar to the way ants organise routes by laying pheromones. Therefore in this piece of work we call the mobile objects ‘ants’, and we speak about ‘pheromone tables’ instead of probability tables. Each node has its own ‘scent’ or ‘pheromone’ that attracts ants with this particular node as a destination. An ant is launched on a source node and update the probabilities in the pheromone tables for this node, while randomly walking according to the probabilities for the node that it is trying to reach.

The behaviour of each individual ant is simple and has a large random component. Some additional mechanisms were added to the system, like congestion dependent delay; increasing influence for younger ants and ants on weak trails; and noise. Section 5.0 will show us how well the ant-based control system performs in solving difficult load balancing problems in telecommunications networks.

A convenient implementation is to arrange that a noise factor of *f* means that at every time step an ant has probability *f* of choosing a purely random path, and probability (1-*f*) of choosing its path according to the pheromone tables on the nodes. The possibly beneficial effects of the addition of noise to ant-based algorithms were noted in [Deneubourg 91]: 'Rather than simply tolerating a certain degree of error, it can even be desirable to deliberately add error where none or little exists.'. In Appendix D some experiments are presented that illustrate some beneficial effects of noise.

3.2.6 General framework for ant-based control systems

The basic principles for ant-based control (ABC) systems are applicable to a wide variety of problems, and can be characterised as follows:

- Ants are regularly launched with random destinations on every part of the system.
- Ants walk randomly according to probabilities in pheromone tables for their particular destination.
- Ants update the probabilities in the pheromone table for the location they were launched from, by increasing the probability of selection of their previous location by subsequent ants.
- The increase in these probabilities is a decreasing function of the age of the ant, and of the original probability.
- This probability increase could also be a function of penalties or rewards the ant has gathered on its way.
- The ants get delayed on parts of the system that are heavily used.
- The ants could eventually be penalised or rewarded as a function of local system utilisation.
- To avoid overtraining through freezing of pheromone trails, some noise can be added to the behaviour of the ants.

3.2.7 Parameters

There is a large number of parameters to tune for this system - choices will be based on experience with a variety of previous simulations:

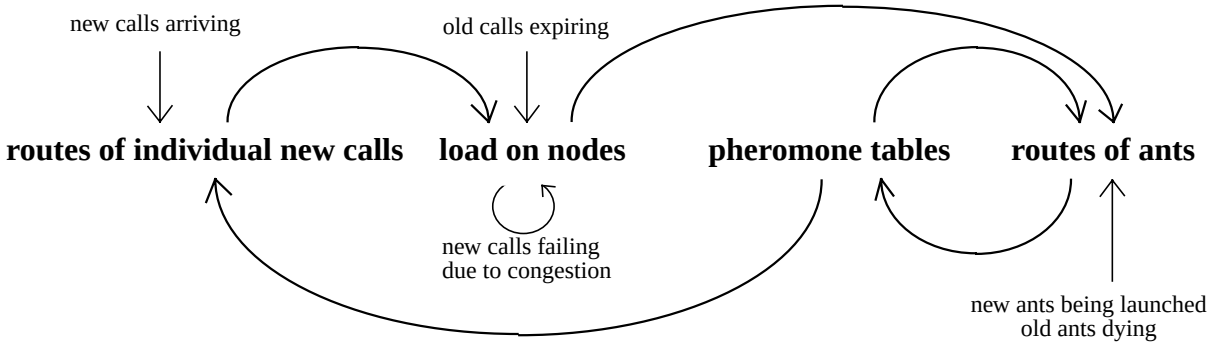
- The speed of the ants.
- The times at which ants get launched.
- The probabilities are updated as explained in Section 3.2.1. The probability change should be a decreasing function of age. A possible formula, satisfying this constraint is: (*age* stands for the number of time steps that passed since the launch of the ant, *c* and *d* are constants):

$$\Delta p = \frac{d}{age} + c$$

- The *delay* in time steps that is given to the ant is an increasing function of the spare capacity *s* of the node. Ants on highly congested nodes should have a more than proportionally bigger delay than ants on lightly used nodes. (The difference between a 5% and 10% spare capacity is more important than the difference between 50% and 55%.) An exponential function would be very suitable (*a* and *b* are constants):

$$delay = \lfloor a \cdot e^{-b \cdot s} \rfloor$$

FIGURE 8. Relationship between calls, node utilisation, pheromone tables and ants. An arrow indicates the direction of influence



3.2.4 Initialisation

A network initialised with random or uniform entries in the pheromone tables will not initially contain any useful information about consistent (i.e. non-circular) call routes, let alone good routes. It therefore makes little sense to examine network performance during this phase. However, even in the absence of calls on the network, the ants will bias towards shortest paths. There are three mechanisms contributing to this:

- The shortest routes will be completed first, and will subsequently direct other ants to their sourcing nodes first.
- Shorter routes involve less branching, so the number of ants travelling over these routes and laying pheromones will be larger than on longer and more branched routes.
- Ants travelling over shorter routes will be younger when they arrive, and will therefore exercise more influence on the pheromone tables.

After a short time the highest probabilities in the pheromone tables of each node will define relatively short routes, and circular routes will have been eliminated (for example after 500 time steps the average route length is typically 4.10, where 4.07 is the minimum possible), and when the routes are sufficiently short, calls can safely be put on the network; subsequent adaptation will then be influenced by any congestion caused by calls. All ant networks were therefore initialised with equal probabilities for neighbour nodes in each pheromone table, and allowed to run for a fixed period before calls were applied. More about initialisation can be found in Section 5.2.

3.2.5 Noise

To overcome the blocking problem and the shortcut problem, we need to avoid ‘freezing’ of the routes in situations that remain static for a long time and then suddenly change. One way of doing this is by adding an exploration probability, or noise, to the random walk of the ants; this will ensure that even apparently useless routes are used occasionally, so that at least some information about them is present in the system to give a head start when a route is blocked (e.g. by extreme node congestion or node failure). Noise might also encourage the more rapid discovery of a better route which suddenly appears.

which are routing ants to the congested node, and allowing the probabilities for alternative choices to increase rapidly.

- since the ants are older than they otherwise would have been when they finally reach the neighbouring nodes, they have less effect on the pheromone tables.

It is of course possible to achieve the second effect alone by increasing the parameter representing the age of the ant without actually delaying the ant; this essentially reduces the effect on pheromone tables of an ant which has passed through a congested node. This has a biological parallel: in some species of ants, those returning from a richer food source tend to drop more pheromone than those from a poor source [Beckers 93]. In the case of my network simulation, however, the combination of delay and age-related penalty seems to be particularly effective. An added advantage of this formulation is that the manipulation of the parameter relating delay to the degree of congestion can be used to control the relative weighting which the system gives to preferring the shortest route (which maximises spare capacity), as against preferring the least congested route.

3.2.3 How calls are routed

Having explained how ants ‘choose’ their routes through the network, let us consider the calls. Calls operate independently of the ants. To determine the route for a call from a particular node to a destination, the largest probability in the pheromone table for this destination is looked up. The neighbour node corresponding to this probability will be the next node on the route to this destination. The route is valid if the destination is reached, and the call is then placed on the network, unless one of the nodes on the route is congested; in that case the call fails to be placed on the network.

In this way, calls and ants dynamically interact with each other. Newly arriving calls influence the load on nodes, which will influence the ants by means of the delay mechanism. Ants influence the routes represented by the pheromone tables, which in their turn determine the routing of new calls. These relationships are illustrated in Figure 8. One needs to realise that the pheromone table by which an individual ant is influenced, is a different table than the pheromone table that will be updated by this ant (see Section 3.2.1). The load on the network at any given time influences which calls can subsequently be placed on the network and which calls will fail; which of course determines the load at a later stage.

that route via their effect on the ants travelling on the opposite route. The system may thus achieve similar effects to the biological bidirectional trail layers, but through an indirect form of interaction. Summarised:

- *Ants travelling from S to D influence other ants that are travelling from any node in the network to S*
- *Ants travelling from S to D are influenced by other ants that are travelling from D to any node in the network.*

This way of directly updating probabilities differs from the way ants lay pheromones, but is functionally equivalent. I feel that using probabilities instead of absolute pheromone quantities helps us to understand the behaviour of the artificial ants better. The tables used here give the probabilities of alternative choices between paths directly, whereas the pheromones of real ants are basically a code that is effectively converted into probabilities by the ant's nervous system.

The method used to update the probabilities is quite simple: when an ant arrives at a node, the entry in the pheromone table corresponding to the node from which the ant has just come is increased according to the formula:

$$p = \frac{p_{old} + \Delta p}{1 + \Delta p}$$

Here p is the new probability and Δp is the probability (or pheromone) increase. The other entries in the table are decreased according to:

$$p = \frac{p_{old}}{1 + \Delta p}$$

Since the new values sum to 1, they can again be interpreted as probabilities. Note that a probability can be reduced only by the operation of normalisation following an increase in another cell in the table; since the reduction is achieved by multiplying by a factor less than one, the probability can approach zero if the other cell or cells are increased many times, but will never reach it. For a given value of Δp the absolute and relative increase in probability is much greater for initially small probabilities than for those which are larger. This has the effect of weighting information from ants coming from nodes which are not on the currently preferred route, a feature which may assist in the rapid solution of the shortcut problem.

3.2.2 Ageing and delaying ants

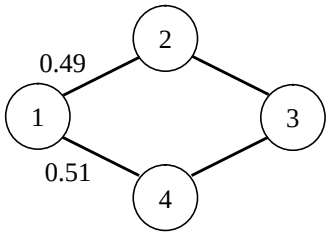
A primary requirement of this work was to find some simple methods of encouraging the ants to find routes which are relatively short, yet which avoid nodes which are heavily congested. Two methods are used. The first is to make Δp, the value used to change the pheromone tables, reduce progressively with the age of the ant. When the ant moves at one node per time step, the age of the ant corresponds to the path length it has traced; this biases the system to respond more strongly to those ants which have moved along shorter trails. The second method, which is related to the first, is to delay ants at nodes that are congested with calls to a degree which increases with the degree of congestion. This delay has two complementary effects:

- it temporarily reduces the flow rate of ants from the congested node to its neighbours, thereby preventing those ants from affecting the pheromone tables

3.2.1 Pheromone tables

The routing tables in the network nodes were replaced by tables of probabilities, which we will call ‘pheromone tables’, as the pheromone strengths are represented by these probabilities. Every node has a pheromone table for every possible destination in the network, and each table has an entry for every neighbour. For example, a node with four neighbours in a 30-node network has 29 pheromone tables with four entries each. One could say that an *n*-node network uses *n* different kinds of pheromones. The entries in the tables are the probabilities which influence the ants’ selection of the next node on the way to their destination node. Figure 7 shows a possible network configuration and a pheromone table. For example, ants travelling from node 1 to node 3 have a 0.49 probability of choosing node 2 as their next node, and 0.51 of choosing node 4. ‘Pheromone laying’ is represented by ‘updating probabilities’.

FIGURE 7. Using ants for network management



The three pheromone tables for node 1:

destination node	next node 2	next node 4
2	0.95	0.05
3	0.49	0.51
4	0.05	0.95

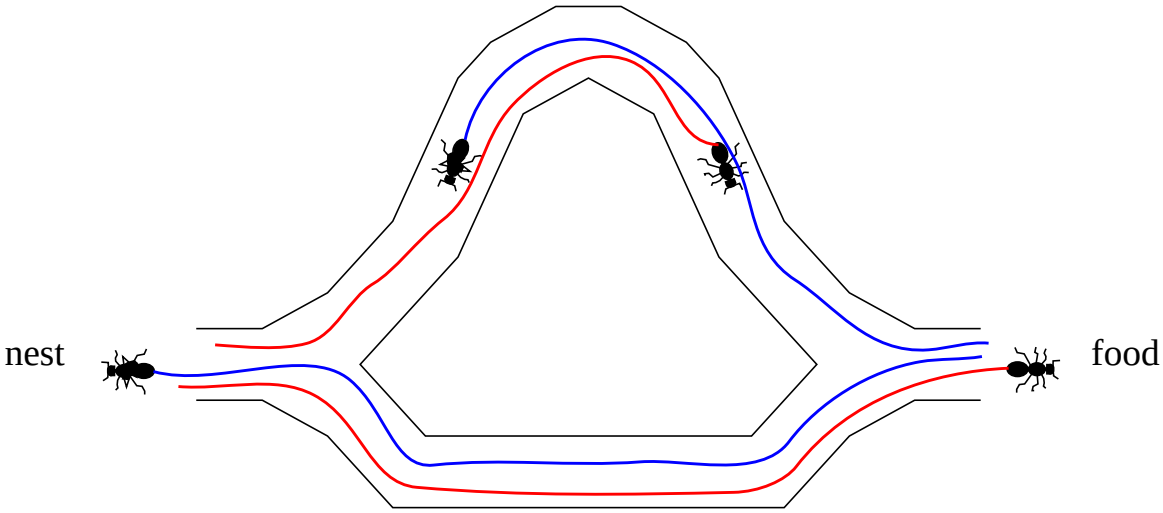
At every time step during the simulation, ants can be launched from any node in the network. Each ant has a random destination node. Ants move from node to node, selecting the next node to move to according to the probabilities in the pheromone tables for their destination node. Arriving at a node, they update the probabilities of that node’s pheromone table entries corresponding to their *source* node i.e. ants lay the kind of pheromone associated with the node they were launched from. They alter the table to increase the probability pointing to their previous node. When ants have reached their destination, they die.

Take as an example an ant in the network of Figure 7 that is launched at node 3 with destination node 2, and has just travelled from node 4 to node 1. This ant will first alter node 1’s table corresponding to node 3 (its source node) by increasing the probability of selection of node 4; it will then select its next node randomly according to the probabilities in the table corresponding to its destination node, node 2. (Note that just for the purpose of illustration this particular ant travelled an ineffective route from node 3 to 2, namely 3-4-1-2. This is possible because ants do not necessarily walk the shortest way, due to the randomness of their walk.)

In this way, ants moving away from their source node can only directly affect those ants for which this source node is the destination node. This is unlike the trails of bidirectional trail laying ants, in which a trail laid in one direction can directly affect ants travelling in either direction. However, the ants which can be directly influenced by an ant travelling from a source node S to a destination node D will include those travelling from D to S; these are the very ants which could be expected to have most influence on ants travelling from S to D, and so ants travelling from S to D may have a strong influence on ants subsequently travelling

other turns right, after a while a situation occurs like that in Figure 6. The lines on the paths represent the pheromone trails. The ants that chose the shorter branch have arrived at their destination, while the ones that chose the longer branch are still on their way. Ants initially select their way with a 0.5 probability for both branches, as there is no pheromone on the paths yet. If there is pheromone present, there is a higher probability of an ant choosing the path with the higher pheromone concentration, i.e. the path where more ants have travelled recently. If at the moment of the situation in Figure 6 other ants arrive and have to choose between the two paths, they are more likely to choose the shorter path, because that is where the concentration of pheromone is higher. This means that the amount of pheromone on the shorter path is more likely to be reinforced again. In this way, a stronger pheromone trail will arise on the shorter path, and so the path will be selected by an increasing proportion of ants As fewer ants choose the longer path, and the existing pheromone slowly evaporates, the trail on the longer path will weaken and eventually disappear.

FIGURE 6. Situation several moments later



Although this is essentially self-organisation rather than learning, ants have to cope with a phenomenon that looks very much like overtraining in reinforcement learning techniques. There are two main issues in this: the blocking problem and the shortcut problem [Sutton 91]. The blocking problem occurs when a route previously found by the ants is no longer available. It can then take a relatively long time for the ants to find a new route, because the trails leading to the blocked route are sometimes very strong. The shortcut problem occurs when a new, shorter route suddenly becomes available. In this case the new route will sometimes not easily be found, because the existing trails are so strong that almost all the ants choose them.

3.2 Artificial ants for network management

How could this trail laying and following behaviour be applied to something like a telecommunications network? And can we overcome the blocking problem and the shortcut problem? This section describes how I implemented an artificial ant population on the network model.

3.0 Ants

The principles that colonies of ants or other social insects use to perform complex tasks, could provide us with a completely new approach for parallel, distributed problem solving e.g. congestion avoidance in telecommunications networks. A type of sign-based stigmergy (see Section 1.3) is used in my network model. It is based on the way ants find short routes from their nest to a food source, and also on the way they select between food sources of different value. They do this by laying and sensing trails of pheromones - specialised chemical substances which are laid in amounts determined by local circumstances, and which by their local concentration subsequently directly influence an ant's choice of route.

3.1 Basic principle of trail laying by natural ants

Depending on the species, ants in nature may lay pheromone trails when travelling from the nest to food, or from food to the nest, or when travelling in either direction. They also follow these trails with a fidelity which is a function of the trail strength, among other variables. Ants drop pheromones as they walk by stopping briefly and touching their gaster, which carries the pheromone secreting gland, on the ground. The strength of the trail they lay is a function of the rate at which they make deposits, and the amount per deposit. Since pheromones evaporate and diffuse away, the strength of the trail when it is encountered by another ant is a function of the original strength, and the time since the trail was laid. Most trails consist of several superimposed trails from many different ants, which may have been laid at different times; it is the composite trail strength which is sensed by the ants. The principles applied by ants in their search for food are best explained by an example as given in [Beckers 92]:

FIGURE 5. Ants have a decision to make

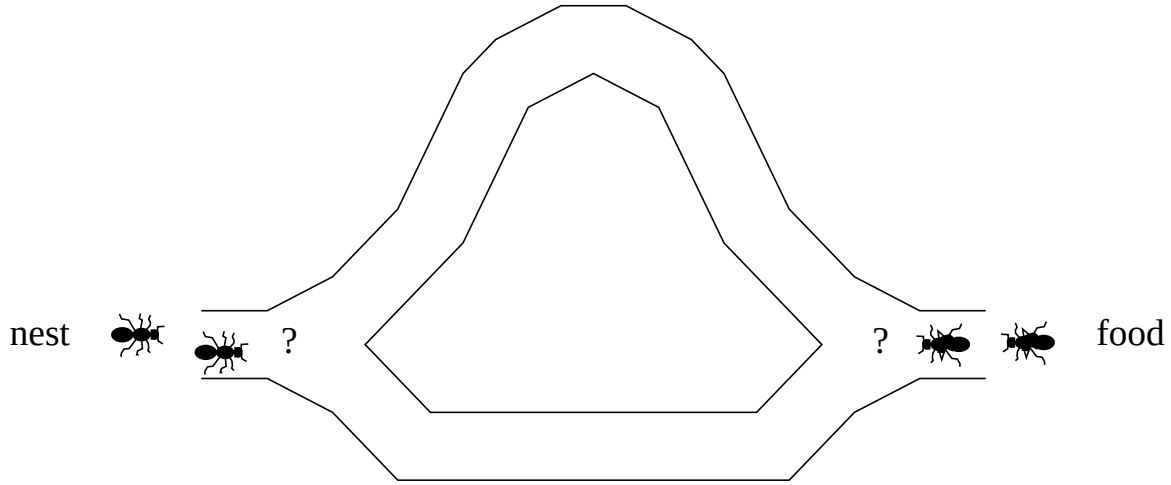


Figure 5 illustrates two possible routes between nest and food-source. Initially, an ant arriving at a T-crossing (choice point), makes a random decision with a probability of 0.5 of turning left or right. Now suppose there are two ants leaving the nest, looking for food, and two ants returning from the food source to the nest. Let the ants be of a type such as *Lasius Niger* which deposits pheromones when travelling both to and from the nest. If one ant from each pair turns left, and the

- The bandwidth of each call, which is the amount of capacity it needs on each node.
- The distribution of the probabilities of each node being the end point of a call.
- The average number of calls generated per time step.
- The average duration of each call.
- The interconnection structure of the network.

The application provides several topologies for testing. My choices for the settings of the parameters will be given in chapter 5.0 ‘Experiments’.

2.6 The network simulation tool

To facilitate seeing exactly what is going on in the network simulation, the traffic in the network is represented visually during run-time by changing the colour of the nodes to indicate their spare capacities. A large number of features to support visualisation and usability have been added to the program, such as:

- displaying the changes with time of one particular route;
- running the simulation step-by-step;
- inspecting every part of the network during the simulation;
- visualisation of the agents;
- monitoring the network by writing statistical measurements on every time step of the simulation to a file.

More details about the implementation and use of the simulation program can be found in Appendix A and Appendix B. One can get an impression of the program by taking a look at the snapshot of Figure 23 on page 73.

of a switched-based telecommunications network, on which the techniques will be tested. Section 2.3 describes this network model; the next two chapters will describe the two different distributed control mechanisms that were investigated.

2.3 The network model

For the testing of distributed load balancing techniques, an application has been written in Smalltalk/VisualWorks™[1]to simulate traffic patterns on a model of a switch-based telecommunications network. Such a network is most naturally represented by an undirected graph. Each node in the graph corresponds to a switching station; the links between nodes correspond to communication channels. In this simulation, a given node will only be linked to a subset of other nodes, usually its geographical neighbours; links are intrinsically bidirectional. The network model used is a graph of *n* nodes, each of which has several attributes:

- A node identifier.
- A capacity. This is the number of simultaneous calls that the node can handle.
- A routing table with (*n*-1) entries, one for each node in the network. Each entry tells us which node is the next node on the route to the destination node concerned.
- A probability of being the end node (either source or destination) of a call.
- A spare capacity. This is the percentage of the capacity that is still available on the node.

The links between the nodes are assumed to have infinite capacity, and infinite transmission speed, so that the geographical distance between linked nodes is of no account. The node capacities will therefore be the only bottlenecks in the network.

2.4 The simulation

Every time step of the simulation proceeds as follows. First, calls that have expired are removed, releasing capacity at the nodes. Then, one or more calls are generated by a traffic generator. These generated calls consist of a source node, a destination node and a duration, measured in time steps. When a call is generated, its route is determined by the current routing tables, and the call is placed on the network, reducing the spare capacity of each node on the route. If there is no spare capacity available on a node on the route of the call, this call is dropped. The expected number of calls that is generated in one time step follows a Poisson distribution, and the duration of each call an exponential distribution. The source and destination nodes are randomly chosen according to their probability of being an end node; this is both convenient and reasonable. Note that the focus of the simulation is on the investigation of new principles, rather than being a simulation of a specific existing telecommunications network.

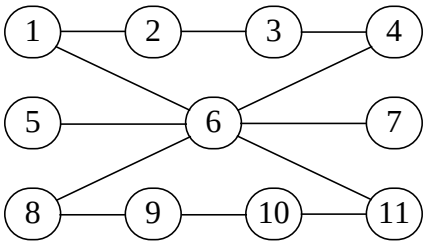
2.5 Parameters

The number of parameters that can vary in this network model is large. This implies that some arbitrary choices have to be made for testing. To mention some of those parameters:

- The capacity of each node.

1. VisualWorks is a trademark of ParcPlace Systems, Inc.

FIGURE 4. Local least cost needs not to be global least cost



The disadvantage of existing adaptive routing schemes is that they often lay constraints on the network. For example British Telecom's Dynamic Alternative Routing algorithm demands that the network is fully interconnected or has a fully interconnected core network. The nodes of a so-called access network are then connected to two 'parent nodes' of the core network. [Gibbens 93]. These constraints are made to make it possible to prove that the algorithm results in optimal network performance, in the sense of a minimum number of call failures.

2.2.3 Centralised and decentralised routing

Adaptive routing algorithms can be centralised or decentralised. When centralised routing is used, there is a central controller somewhere in the network. The network nodes send periodically information about their states to the controller. According to this information, the controller calculates better routes from every node to every other node in the network, and sends the resulting routing tables to the nodes.

Centralised routing appears to be very attractive. As the controller has complete information about the entire network available, it can make seemingly perfect decisions. However, controlling large and complex distributed systems by means of a single central controller has several disadvantages, too. The controller usually needs current knowledge about the entire system, necessitating communication links from every part of the system to the controller. These central control mechanisms scale badly, due to the rapid increase of processing and communication overheads with system size. Failure of the controller will often lead to failure of the complete system. There is the additional practical commercial requirement that centrally controlled systems may need to be owned by one single authority. Further, the nature of distributed systems like these is highly dynamic, complex and stochastic, and their behaviour can neither be predicted nor explained by reducing it to a single central controllable factor.

A good decentralised control mechanism will not have the problems mentioned above. It may consist of many simple, interacting entities instead of one very complex program. The field of artificial life has given me inspiration for such a mechanism that will be completely distributed, and highly adaptive to changes in the network and traffic patterns (this mechanism is explained in chapter 3.0). This solution makes use of the distributed processing capability already inherently present in the network in the form of network nodes. The distributed nature of such an approach may make the system very robust against failures of individual control entities.

Two different decentralised approaches are presented in the next chapters. Both techniques are based on computational processes which move from node to node in the network. An application has been written to simulate traffic patterns on a model

Another possibility is called packet switching. Here a message that is sent from one computer to another may be decomposed into packets. No fixed connection is made, and the packets with the same source and destination may eventually travel from switching exchange to switching exchange over different routes.

2.2.2 Non-adaptive and adaptive routing

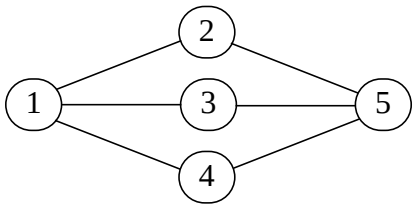
Routing algorithms can be divided into two main-classes:

- Non-adaptive routing algorithms. Here routing decisions are made off-line and are downloaded to the switching nodes of the network.
- Adaptive routing algorithms. Here the routes dynamically adapt automatically to changes in network load and network failures.

A non-adaptive algorithm could for example make the routes to be the shortest routes from every node in the network to every other node, according to a user-defined cost function. These routes are then downloaded into tables of the switching elements. The routes are then only adapted when a node or link is added to the network. An adaptive algorithm could for example use a cost-function to update the routes on moments of bad network performance. To find the least-cost route of packets or calls at the moment of sending or initialisation is practically an impossible task.

Why do we need adaptive network routing? An example network to illustrate the advantages of adaptive routing algorithms is shown in Figure 3. Suppose that the nodes in the graph represent network nodes, and the lines represent network links. If all links and all nodes have the same capacity, and if packets are mainly sent between nodes 1 and 5 - then any fixed configuration of routing tables would leave at least one of the three available routes unused (we do not consider probabilistic routing here). An adaptive routing scheme would perform much better, as the packets could then be distributed among the three available routes.

FIGURE 3. Example network to illustrate importance of adaptive routing schemes



Another problem occurring with fixed least cost routing algorithms is shown by the network of Figure 4. Suppose now that connections are mainly made between nodes 1, 5, 8, (the left-hand nodes) and nodes 4, 7, 11 (the right-hand nodes). Suppose now that the cost is defined as the number of links and nodes of a route. The least cost route for all routes would go via node 6. However, in this way node 6 will get highly congested, and therefore it might be useful to chose alternative routes that go over more nodes. The solution is relatively simple in this network; but in larger networks it gets very hard to find low cost routes that do not cause congestion (take a look at the network graph of Figure 13 on page 37 for example). Adaptation of a route because of congestion might cause congestion elsewhere. To minimise lost calls the loads on the nodes should be as balanced or equally distributed as possible.

2.0 A Network Simulation to Investigate Distributed Load Balancing Techniques

2.1 Introduction

As pointed out in the previous chapter, the notion of complex collective behaviour emerging from the behaviour of relatively simple units and the interactions between them, is fundamental to the field of artificial life and behaviour-based AI. The growing understanding of such systems offers the prospect of creating artificial systems which are controlled by such emergent collective behaviour; in particular, I believe that the exploitation of this concept might lead to completely new approaches for the management of distributed systems, such as load balancing in telecommunications networks.

What is load balancing? For economic and commercial reasons, such networks are equipped not with a level of equipment which will guarantee successful call connection under all possible circumstances, but with some lower level which will give acceptable performance under most conditions of use. If there is some significant change in the conditions - for example, if the total call volume at any time is unusually high, or if some particular location is suddenly the origin or destination of an unusually large volume of calls - then these capacity limitations might lead to the system failing with calls unable to be connected.

Calls between two points are typically routed through a number of intermediate switching stations, or nodes; in a large network, there are many possible routes for each such call. It is thus possible to relieve actual or potential local congestion by routing calls via parts of the network which have spare capacity. Load balancing is essentially the construction of call-routing schemes which successfully distribute the changing load over the system and minimise lost calls. Of course it is possible to determine the shortest routes from every node to every other node of the network. In this way the average utilisation of nodes will be minimised, but this is not necessarily the ideal way to avoid node congestion, as this has to do with how the traffic on the network is distributed.

2.2 Some considerations on network routing

There is a variety of load balancing or routing strategies available. Most of these strategies are closely related to particular properties of specific networks. In this section some considerations are mentioned about different kinds of networks and algorithms. More about routing can be found in [Tanenbaum 88] and [Schwartz 94].

2.2.1 Circuit Switching and Packet Switching networks

One of the most important properties of a network is whether it is circuit switched or packet switched.

When for example a telephone call is made, the network switching devices look for a connection from the source of the call to its destination. At the start of a phone call, the route is initialised according to the routing tables in the network switching centres (or nodes), and a fixed number of resources is allocated. Such resources can be bandwidth on the network links, and switching capacity. These resources remain allocated for the entire duration of the call, and are made available again as soon as the call is finished. This way of making connections in a network is called ‘circuit switching’.

- Strong emphasis on reaction and adaptation.
- Bottom-up design.

To summarise these differences, basically the following distinctions can be made:

- focus on explicit symbolic knowledge versus focus on resulting behaviour;
- one single high-level activity versus multiple lower level activities;
- user-driven computation versus autonomous operation;
- learning (there is a ‘knower’) versus (emergent) adaptation.

One can conclude that both approaches study different classes of problems and use different techniques to solve them. One should not forget however, that knowledge-based AI is a decades old discipline, and has clearly proven its applicability to solve particular problems, where this still remains to be seen for behaviour-based AI, or Alife.

From the intelligent autonomous agents point of view, there exists a continuum from ALife agents to what is generally referred to as distributed AI agents. ALife agents tend to be relatively simple, obtaining complex (intelligent) behaviours as an emergent property of their collective behaviour. Distributed AI agents generally are more complex in computation and communication, and often use symbolic models of the world they reside in.

Despite the differences between the experience with the two disciplines (thirty years of knowledge-based AI, large community of researchers, versus ten years of behaviour-based AI in a small community), I believe that artificial life provides an exciting new approach to do artificial intelligence research, and in the longer term more problems will be solved thanks to this field of study, that otherwise would not have been solved with reductionistic top-down approaches commonly used in more traditional artificial intelligence research.

neurons. Neural networks are typically used for pattern recognition and control tasks both within and outside the field of artificial life. Within artificial life neural networks are often used as controllers for animats. An alternative learning technique regularly used in artificial life is to evolve neural network controllers for animats (both the weight-vectors and the interconnection structure) with a genetic algorithm, see for example [Belew 92] and [Collins 90].

1.8 Artificial Life and Artificial Intelligence

Where traditional artificial intelligence (AI) generally tries to solve problems top-down, with symbolic representations and reasoning techniques, artificial life adopts a bottom-up approach without centralised control. However, the exact location of the border between the fields of artificial life and artificial intelligence is unclear. There is an overlap between the two fields, and one could also consider artificial life as a subfield of AI - there are some topics which are considered to belong in both fields, for example intelligent autonomous agents, neural networks and genetic algorithms.

The typical kind of AI studied by artificial life researchers could be classified as behaviour-based AI, as opposed to the more traditional knowledge-based AI [Maes 93]. Rodney Brooks (see Section 1.5.1) is one of the main originators of behaviour-based AI, although the picture he presents is restricted towards robotic forms of intelligence.

Knowledge-based AI systems can be characterised as follows [Maes 93]:

- They are isolated, and have specialised, advanced competences.
- They deal with one problem at the time.
- They use declarative, static knowledge structures.
- The main concern is how to model knowledge, and reasoning and planning mechanisms
- They are generally not adaptive to changing situations.
- The problem domain, and the problem are defined in symbolic terms by users. The returned solution is also symbolic, and has then to be implemented by the user in that domain.
- No direct interaction with the problem domain. The connection with the environment is via the user; or via a separate perception and execution module.

Behaviour-based AI systems are typified by the following properties:

- They have multiple lower level competences.
- The system is situated in its environment. It uses ‘the world’ as its own model. It is directly connected to its problem domain through sensors and effectors.
- The studied problem domains are dynamic and complex, and there is a limited time to act.
- Typically they are autonomous and need no human guidance to operate. They need no explicit goal formulation from the user.
- The emphasis is on the resulting behaviour of the system, rather than on knowledge.
- Results can be emergent and need not to be attributed to particular internal structures.

- Much effort might go into solving problems that do not occur in the real world.
- The simulation of the dynamics of the real world, and real-world sensing and actuation, is very hard to simulate. Therefore it is likely that programs which work well on simulated robots will completely fail on real robots.

Nevertheless, several attempts have been made to use simulated evolution to create real-world robot controllers. Two examples can be found in [Nolfi 94] and [Cliff 93].

1.6 Evolutionary learning techniques

Genetic algorithms and genetic programming techniques are inspired by Darwin's evolution theory, and based on the principle of 'survival of the fittest'. They are used to solve problems of which solutions can be represented as a string of bits, or other symbols. A genetic algorithm is a stochastic search technique which allow fitter solutions to 'survive' and to be used to make new (child-) solutions. In this way new generations of solutions are made. After a number of generations, a sufficiently 'fit' solution to the problem is chosen. Genetic programming results in computer programs that are evolved according to a similar principle. Next to the evolution of artificial creatures (animats), genetic algorithms and genetic programs are used to optimise general problems that are outside the field of Artificial Life. There are different ways to determine the 'fitness' of a solution: Three common ways are:

- to let the user chose which solution will be used for the next generation. Fitness is determined by the users' happiness with the solution. An example of this form of evolution can be found in Dawkins' Blind Watchmaker [Dawkins 87].
- to measure fitness according to an explicit fitness function. At present this is the most commonly used way to measure fitness.
- to let solutions (or creatures) survive in an artificial world. Fitness is a matter of survivability. An example of this philosophy is Tierra, an environment in which organisms consisting of series of machine code compete for resources (memory) and reproduce [Ray 92].

Evolutionary techniques make up a field in their own right, and are not necessarily a subfield of artificial life. The argument for the use of evolutionary techniques is often not only their usefulness, but also their biological plausibility. Researchers expect that simulated evolved creatures have similarities with real animals, and teach us something about nature.

More about genetic algorithms and genetic programming can be found in [Koza 93]. Examples of genetically evolved 'animats' in a graphically represented virtual world can be found in [Sims 94] or [Reynolds 94].

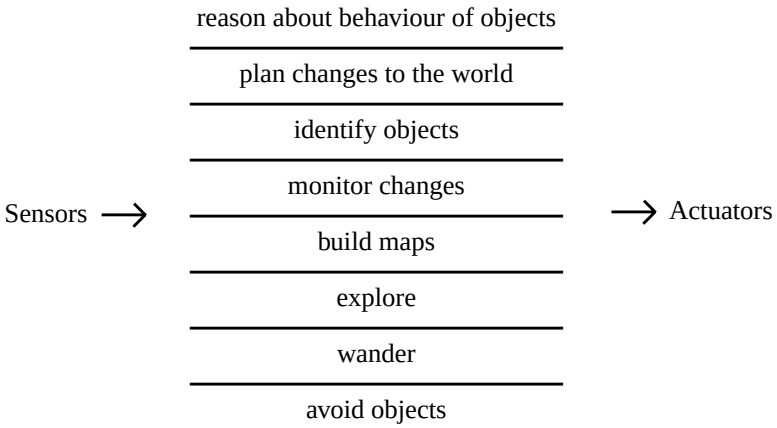
1.7 Artificial neural networks

Other techniques that are often used for artificial life come from the field of artificial neural networks. They are inspired by networks of neurons in the human brain. A typical neural network consists of an input-layer of neurons, one or more hidden layers, and an output-layer. The neurons of the adjacent layers are connected via 'artificial synapses'. The strength of these connections is typically represented by a weight-value. Learning techniques like the backpropagation algorithm [Hecht-Nielsen 91] make the neural network able to approximate functions. These functions are 'learnt' by means of changing the weights of the connections between

system. These components could be competing modules, that are able to directly produce behaviours and are concurrently present on a system.

In the spirit of these ideas, Brooks proposed a radically different architecture for building mobile agents, based on decentralized, adaptive control. His idea was to decompose the system into modules that implement different levels of task-achieving behaviours, or levels of *competence.* This is illustrated by the picture below, as given in [Brooks 86]:

FIGURE 2. Decomposition based on task-achieving behaviours in the Subsumption Architecture.



When building a mobile robot in this way, one starts with the lowest layer. In the example of Figure 2 this layer provides the ability to avoid objects. This layer is debugged thoroughly, and the robot will be able to avoid objects. After this layer has proved to function well, it will not be altered anymore, and the second level of behaviour is implemented. This level is able to suppress the normal data flow of the first level by examining data of the first level and injecting data into it. This level is therefore able to *subsume* the role of the lower level when it wishes to take control and therefore he calls this architecture the 'Subsumption Architecture'. Imagine the situation that there are no obstacles in the neighbourhood of the robot. Then the 'wander'-layer might want to take over control of the 'avoid-objects'-layer, i.e. *subsume* the role of the 'avoid-objects'-layer. After thoroughly debugging the newly added level, and 'freezing' it, next levels can be added in the same way. These layers only need a low bandwidth to communicate with each other. The whole system is much less computationally expensive than traditional systems. When testing a robot based on the Subsumption Architecture, seemingly complex behaviours can be observed, due to the relatively quick alterations of different levels taking over control. In [Steels 94] an alternative approach is presented in which different levels do not subsume each other, but are competing processes in an on-line evolutionary process.

1.5.2 The simulation debate

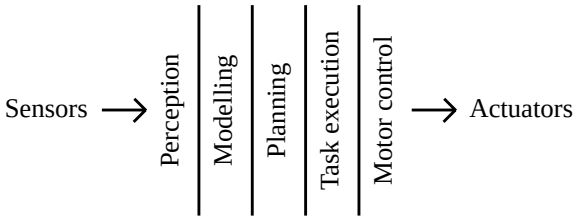
There is a debate going on whether it is possible to develop controllers for autonomous agents in simulated environments or not. As the section above indicates, Rodney Brooks is strongly opposed to making controllers for physically embodied agents by means of simulation. In [Brooks 91] he gave the following reasons:

novel approach to build artificial intelligent controllers for mobile robots, which also has its influence outside the field of robots [Brooks 86].

A traditional robot control system consist of a sequence of functional units of layers from sensors to actuators:

FIGURE 1.

Example of a decomposition into functional modules according to the traditional approach to build mobile robots.



In this traditional architecture, a robot perceives the world through its sensors and analyses this data using a variety of pattern recognition techniques. Then a model of its physical environment is built, and a plan is made based on this model. According to this plan the motor control routines are called to effect the actuators. As pointed out in [Brooks 91], these robots are very computationally expensive and fail when placed in unanticipated situations. The world in which these robots function always needs to be specially engineered to enable the robots (or agents) to build internal geometric models of it.

Brooks states that these supposedly artificially intelligent systems are incapable of even very simple activities that humans normally do all the time in the real world. He proposed an alternative set of guidelines to do artificial intelligence research in [Brooks 91]. The key-ideas he identified were situatedness, embodiment, intelligence and emergence.

- *Situatedness*. Rather than solving problems in modelled worlds, an agent should use the world as its own model and directly refer to its sensors.
- *Embodiment*. According to Brooks, only physically embodied agents are fully validated as able to deal with the real world, and really give ‘meaning’ to the processing that is going on in its systems.

Brooks then claims that the best way to obtain intelligent systems is by building real-world robots. This statement has been criticized in an article called ‘Intelligence without Robots’ by Oren Etzioni ([Etzioni 93]). He states that software agents (or softbots, as he calls them) could also exhibit intelligent behaviour in complex software-environments like operating systems, databases or telecommunications networks. This misunderstanding could probably largely be solved by interpreting embodiment as being made of the same structures or material as the (complex) environment an agent works in. The other two other key-ideas presented by Brooks were:

- *Intelligence*. As perception and mobility will be the most difficult problems to overcome, one should study simple animals as a bottom-up model for intelligence. Brooks’ theory is that intelligence is ‘determined by the dynamics of interaction with the world’.
- *Emergence*. Intelligent behaviour will be achieved as a result of the interaction between many very simple components. There is no ‘seat of intelligence’ in a

automaton. The states of the cells of a class IV automata can only be computed by running the cellular automata themselves.

According to Langton in [Langton 90b] the complex dynamics arising from class IV automata suggests that there is a dynamics of information which has taken control over a dynamics of energy. He showed this by creating a λ-parameter, which stands for the amount of information being exchanged in a system. Increasing this value in a CA, leads from stable to periodic to chaotic behaviour. However, there was a small region between the values yielding periodic and chaotic behaviours, where complex patterns arised. This appeared to point out the optimum level of information exchange which makes life possible. The phrase 'the edge of chaos' was introduced to indicate this region.

An interesting example of such a 2-dimensional class IV cellular automata is John Conway's game of life[1]: Each cell can occupy two states; ON and OFF. Each cell has eight neighbours, and the rule-set is as follows:

- If the number of neighbours of the cell that is ON is two, then the cell remains in its present state
- If the number of neighbours of the cell that is ON is three, the cell will be ON in the next time step
- Under all other conditions the cell is OFF.

When observing a cellular automata with these simple rules, one can however observe many complex patterns like glider-objects, stable forms, and seemingly chaotic phenomena. Several internet web-sites can be found with demonstrations of 'the game of life'.

Cellular automata provide another illustration of emergent collective behaviour: The patterns arising from a class IV cellular automaton cannot be predicted in advance, just by looking at the rule set, however simple this rule set might be.

1.5 Autonomous agents

Another area that is associated with artificial life is autonomous agents. Agents are computational systems that interact with dynamic and unpredictable environments. They observe the environment through sensors and produce effects in the environment by executing motor commands. [Russell 95]. An autonomous agent needs no human intervention during its life time.

Autonomous agents inhabiting the physical world can be considered as robots. Autonomous agents can also inhabit digital environments like computers and networks, in which case they are often called softbots [Etzioni 93]. Another word that is often used in the artificial life community for an autonomous agent is 'animat', as proposed in [Wilson 90], to stress the analogy with real animals.

1.5.1 The Subsumption Architecture

Rodney Brooks takes a special place in the area of autonomous agents. In 1986, he wrote an influential paper in which he introduced the Subsumption architecture, a

1. The description is adapted from www-link: http://www.cs.vu.nl/~jaharkes/vugrail/cnri/demo/life/

- preferentially exploiting the richest available food source.

These behaviours emerge from the interactions between large numbers of individual ants and their environment. In many cases, the principle of stigmergy is used. Like other insects, ants typically produce specific actions in response to specific local environmental stimuli, rather than as part of the execution of some central plan. If an ant's action changes the local environment in a way that affects one of these specific stimuli, this will influence the subsequent actions of ants at that location. The environmental change may take either of two distinct forms. In the first, the physical characteristics may be changed as a result of carrying out some task-related action, such as digging a hole, or adding a ball of mud to a growing structure. The subsequent perception of the changed environment may cause the next ant to enlarge the hole, or deposit its ball of mud on top of the previous ball. In this type of stigmergy, the cumulative effects of these local task-related changes can guide the growth of a complex structure. This type of influence has been called sematectonic [Wilson 75]. In the second form, the environment is changed by depositing something which makes no direct contribution to the task, but is used solely to influence subsequent behaviour which is task related. This sign-based stigmergy has been highly developed by ants and other exclusively social insects, which use a variety of highly specific volatile hormones, or pheromones, to provide a sophisticated signalling system. Some of the above behaviours have been successfully simulated with computer models, using both sematectonic stigmergy [Bonabeau 94], and sign-based stigmergy [Stickland 92] and also on robots [Beckers 94].

One can view stigmergy together with learning as the two key mechanisms in obtaining adaptive behaviour, although applied research has predominantly focused on the latter one so far. However, I think stigmergy is an approach with a large number of potential applications. Section 3.0 will focus on an application of sign-based stigmergy based on the way ants find short routes from their nest to a food source, and also on the way they select between food sources of different value. The way ants organise these routes has inspired me to investigate a new approach for congestion avoidance in telecommunications networks.

1.4 Cellular Automata

Cellular Automata are often used as a modelling framework for artificial life simulations. They are mathematical models containing many simple, identical components that together are capable of highly complex behaviours. A Cellular Automaton is a D-dimensional lattice with a finite-state automaton at each cell. The input of each cell consists of the states of the cells within a finite, local region of the lattice. The automaton has a set of rules which map each combination of its current state and the state of the cells in their neighbourhood to a new state. All cells are updated synchronously. In this way the state of the entire lattice advances in discrete time steps. One of the interesting properties of cellular automata is their capability to self-reproduce, as shown by John von Neumann in 1966.

Wolfram showed in [Wolfram 84] that one dimensional and perhaps all cellular automata can be divided into four classes (from I to IV), exhibiting respectively fixed, periodic, chaotic and complex behaviours. These classes can be both recognised qualitatively by looking at the patterns, and quantitatively by using statistical measures of entropy. For the first two classes, and probably the third, there exist a simple algorithm to compute the state of each cell of a cellular automaton after any finite number of time steps, without having to run the cellular

In artificial life simulations global patterns can emerge as a result of the behaviour of individual entities and the interactions amongst them. Artificial life adopts a bottom-up approach in which only the behaviour of lower-level units is programmed, and 'higher level' collective behaviours are obtained from the interactions between these units [Kawata 94]. This higher-level behaviour is generally referred to as 'emergent behaviour' or 'emergent functionality'. Emergent behaviour is one of the most important notions in the field of artificial life.

1.2.1 Swarms

A major area of interest in artificial life research has been the simulation of swarms of animals. In these swarms global patterns emerge as a result of the behaviours of individual entities, where the entities are animals. Examples of simulations of such emergent collective behaviour include: bird-flocking [Reynolds 87], fish-schooling [Zaera 95], nest-building by social insects [Bonabeau 94] and foraging of ants.

In the first instance people have a strong tendency to interpret these behaviours as being the result of a central controlling force. However, most flocks of birds or schools of fish do not have leaders. Instead, the individual animals behave according to relatively simple rules that could easily be explicitly programmed, or evolved with a genetic algorithm (see Section 1.6), and react to their environment and fellow-animals near-by [Resnick 94]. The seemingly complex flocking behaviour can be a result of simple rules of interaction between the animals. Flocking simulation has some practical applications in computer graphics for movies, video games, and screen savers.

1.3 Stigmergy

An important collective behaviour mechanism that makes insects perform well in a large variety of tasks is called stigmergy [Grassé 59]. Stigmergy is a form of indirect communication through the environment. Many different examples of applications of this principle can be observed in ants, but many other social insects use it too, including wasps, bees, and termites.

1.3.1 Ants

I will go deeper into the concept of stigmergy in the context of ants, as ants provide the main inspiration for the research topic presented in chapter 3.0. Individual ants are behaviourally very unsophisticated insects. They have a very limited memory and exhibit individual behaviour that appears to have a large random component. Acting as a collective however, ants manage to perform a variety of complicated tasks with great reliability and consistency. A few examples of collective behaviour that have been observed in several species of ants are [Hölldobler 94], [Franks 89]:

- regulating nest temperature within limits of 1ºC;
- forming bridges;
- raiding particular areas for food;
- building and protecting their nest;
- sorting brood and food items;
- cooperating in carrying large items;
- emigration of a colony;
- complex patterns of egg and brood care;
- finding the shortest routes from the nest to a food source;

1.0 Artificial Life

1.1 Introduction

The study of living systems is traditionally done in the field of biology. For obvious reasons, research in this area restricts itself to examples of life as we know it on earth, which happens to be grounded on carbon-based molecules like RNA and DNA. Christopher Langton together with biologists and philosophers before him recognised that the property of an object to be alive is not dependent on its matter, but on its dynamic form [Langton 87]. On earth these dynamic forms all seem to occur in systems based on these RNA or DNA molecules, but there is no reason to believe that these dynamic forms can not be created in other media (e.g. memory of computers) as well. Creating life-as-it-could-be as an alternative to life-as-we-know-it, is the only way to really understand the general properties of life. Langton posed the term *artificial life* for "the study of man-made systems that exhibit behaviours characteristic of natural living systems". It views life as a property of the organisation of matter, rather than a property of the matter itself.

In 1987 people from a wide variety of research areas (biology, computer science, mathematics, philosophy) came together to hold the first of a series of conferences on Artificial Life. Some subjects treated at that workshop were cellular automata, the origin of life, mathematic modelling of growing systems, evolutionary computing, and simulating swarm-like behaviour in animal flocks. Since that time an increasing number of researchers became interested in the field, or realised that their activities were part of it. The fifth conference in this series took place in Japan in May 1996. Since 1990 there is also a closely related series of conferences called "Simulation of adaptive behaviour (From Animals to Animats)", and a number of european artificial life conferences. The two most important journals in the field are "Artificial Life", and "Adaptive Behavior", both published by the MIT Press. The latter has more emphasis on simulation of biological and ethological phenomena.

Next to the possibility of obtaining better insights about life and intelligence, the field of artificial life could possibly provide new techniques that will solve problems in a radically different way to the traditional mathematical or symbolic problem solving techniques. It is a general belief in the artificial life community that the systems based on artificial life techniques will be better able to perform satisfactory in increasingly complex environments, just like real living systems. Such environments could be the real world, a virtual reality, or a virtual environment like a computer network. A practical advantage is also that artificial life techniques turn out to be extremely suitable for implementation on parallel computers.

1.2 Emergent collective behaviour

Natural living systems have complex characteristics that emerge from the interactions between relatively simple units and their environment. Examples of such emergent characteristics can be found at almost every behavioural level in nature. To mention just a few: Most animals consist of complex organs, each consisting of a collection of simple cells. An intelligent brain consists of relatively much simpler neurons, and the growing of all parts of a living entity are influenced primarily by genes, which are in fact just chains of simple molecules. In this way every organism is composed of much simpler entities, and as Aristotle stated it: 'the whole is more than the sum of its parts'.

Table of contents

Abstract

This work presents a novel method of achieving load balancing in telecommunications networks. A survey is given of the field of artificial life, that provided the main inspiration for this method. The key ideas of this field and its relation with other artificial intelligence techniques are explained. Following an overview of the problems of load balancing in networks, a simulated network model is described. This network models a typical distribution of calls between nodes; nodes carrying an excess of traffic can become congested, causing calls to be lost. To solve these congestion problems an approach is presented, which is modelled on the trail laying abilities of ants. Artificial ants move across the network between randomly chosen pairs of nodes; as they move they deposit simulated pheromones as a function of their distance from their source node, and the congestion encountered on their journey. They select their path at each intermediate node according the distribution of simulated pheromones at each node. Calls between nodes are routed as a function of the pheromone distributions at each intermediate node. The performance of the network is measured by the proportion of calls which are lost. The results of using the ant-based control (ABC) are compared with a previously proposed approach using mobile agents. In this approach the network populates parent agents which launch load agents if they observe congestion problems in the network. The load agents make quick routing updates in an algorithmic way according to the current situation. The ABC system is shown to result in fewer call failures than the other method, due to a number of attractive features, which make the system able to reorganise routes based on a combination of network statistics and temporary situations. To carry out this research, a graphical network simulation tool was written.

disadvantages of ant-based control techniques are pointed out and compared to a ‘least cost’ method like the mobile agents.

The implementation of the network simulation tool that was written and used for this research project is described in Appendix A, and Appendix B goes into the use of this program. More details on the experiments from chapter 5 can be found in Appendix C. Appendix D shows some results on experiments of the investigated load balancing techniques on simpler networks.

The research project of this thesis resulted also in the paper ‘Ant-based load balancing in telecommunications networks’, submitted to the journal ‘Adaptive Behavior’, special issue on collective intelligence. That paper was co-authored by Owen Holland, Janet Bruten and Leon Rothkrantz.

Acknowledgments

I would never have been able to succeed without the support of the following people. I would like to thank:

- My HP manager Janet Bruten, for guiding me around in HP, for being always there to help me with problems, and for having many fruitful discussions. She has been an excellent manager.
- Owen Holland from the University of the West of England, for making me at home in the ant-world, and for discussing a lot of new ideas and his co-operation and help in writing them down. I enjoyed working with him very much.
- My Delft University supervisor Leon Rothkrantz, for trusting me to go abroad, for keeping the link with my faculty in Delft, and being on the other side of the copper wire to answer my questions.
- The people from HP Labs for all their co-operation and help, and for having been a pleasant community to work in.
- All friends I made in Bristol for having a lot of fun during this traineeship period.

Ruud Schoonderwoerd[1],
Bristol, 30 May 1996

1. email: `R.Schoonderwoerd@twi.tudelft.nl`, or `ruud@hplb.hpl.hp.com`

commercially exploited at the moment, there is no reason why they will not help us in solving many problems.

Computer simulations of natural phenomena are typically studied in the field of artificial life, and involve very little explicit modelling or reasoning. A difficulty with artificial life is that most work in the area focuses on simulation of life-like behaviours, with as a goal to understand life rather than finding practical applications. However, this thesis is written in the belief that artificial life could also provide new techniques for control of complex systems.

An example of such a complex system is a telecommunications network. Truly optimising the routes of traffic in the enormous telecommunication systems nowadays is a practically impossible task. Local least cost algorithms are not satisfactory anymore as they do not imply global least cost. Instead we could try to find out how nature deals with such complexity, and base new solutions on these examples. My choice was to have a closer look at ants. The interesting thing about an ant colony is that it is often seen by biologists as one super organism that exhibits collective intelligence. The study of the trail laying behaviour of ants might lead to a new approach for organising routes in networks.

During the research project, I tried to achieve the following goals:

- Obtain insights and give an overview of the field of Artificial Life;
- Formulate a decentralised algorithm to balance the load in telecommunications networks, modelled on the trail laying and following behaviour of ants;
- Implement a network simulation system to investigate the properties of this algorithm;
- Compare it with other (distributed) methods.

About this thesis

This thesis is organised as follows:

Chapter 1 provides a brief introduction into the key-ideas of artificial life, laying some stress on the subjects important for this research project, like the concepts of stigmergy and subsumption;

Chapter 2 goes into the problems of load balancing or routing in networks and describes a simulation model used to investigate new distributed approaches;

Chapter 3 describes the trail laying mechanisms of ants. Hereafter a new technique to organise the routes in telecommunication networks is proposed, based on these principles. This ant-based control technique is implemented in the network model of chapter 2;

Chapter 4 describes my implementation of another method, based on mobile software agents previously proposed by British Telecom. Some corrections and improvements to the original algorithm were made;

Chapter 5 contains the complete description the design of the experiments to compare both methods with a fixed routing scheme. An analysis is given of the adaptation to variations in traffic patterns by both methods, and the advantages and

Preface

This thesis is the result of a research project at Hewlett-Packard Laboratories Bristol, performed from 1 September 1995 until 31 May 1996. It is submitted for the title of ‘Ingenieur (ir.)’ at Delft University of Technology, Faculty of Technical Informatics and Mathematics, Knowledge Based Systems group, headed by Prof. dr. H. Koppelaar.

Hewlett-Packard Laboratories Bristol

Hewlett-Packard is one of the leading companies in the areas of measurement, computing, and communications - a triad called MC-squared. Hewlett-Packard Laboratories Bristol (HPLB) is HP’s leading technological research laboratory in Europe. It was founded in 1984 and soon became the company’s largest research centre outside HP’s California headquarters. It shares a site with one of HP’s manufacturing divisions.

HPLB employs around 250 highly qualified researchers, forming a multinational community with contacts spanning HP, its customers and universities throughout the world. It has a sophisticated computing and communications infrastructure offering state-of-the-art software development platforms and hardware prototyping facilities. Every employee or student is equipped with a personal workstation connected to HP’s corporate networks and Internet access. Researchers work in so-called cubes - office spaces divided by low walls - providing an open environment facilitating communication among each other.

Some years ago research at HPLB had a strong focus on Knowledge Based Systems and other Artificial Intelligence techniques. In recent years this focus shifted towards the areas of Networks, Communications and Personal Systems. People at HPLB are from a wide variety of backgrounds.

At the moment there are three laboratories in Bristol, each consisting of roughly four departments of around twenty people. Each department works on several projects - I was based in the Intelligent Networked Computing Laboratory (INCL), in the Mediated Communication Systems Department (MCSD), on the Artificial Life technology watch project headed by Janet Bruten. This is a long-term research project based on the idea that Artificial Life may offer a new approach to the control of large distributed systems. Several students have visited and performed research projects on for example the simulation of fish-schooling behaviour by evolved neural nets; channel assignment in mobile communications by means of classifier systems; and simulation of bird flocking.

Research objectives

There are basically two areas in computer technology which traditionally took their examples from ‘natural intelligence’ to be able to deal with complex problems. One of them is the area of neural networks, which are based on the principles of the neurons in a (human) brain. The other one is the area of knowledge based systems: they are based on human reasoning techniques. Human reasoning however is a subject that is extremely difficult to grasp, as it is implemented by a very complex system itself - the human brain. I suspect that there are much simpler, but also useful mechanisms to be found in nature. Although these principles might not be

Collective Intelligence for Network Control

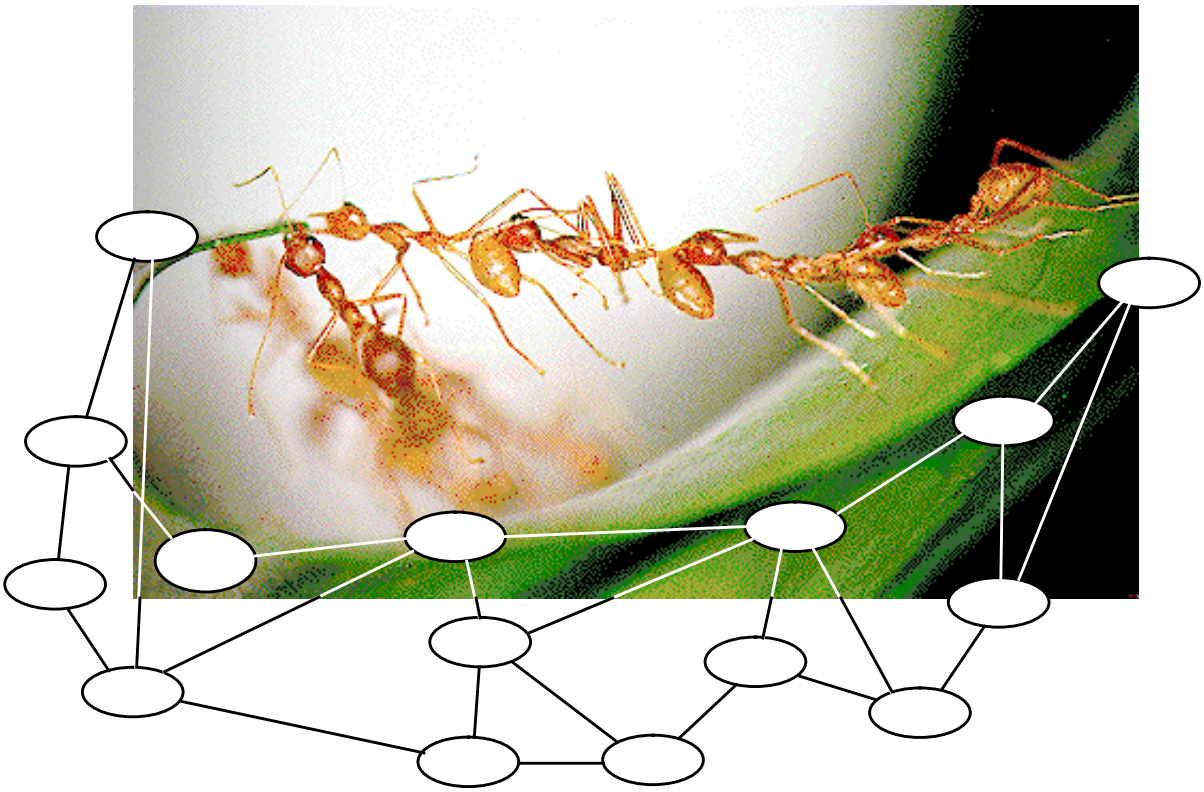
TU Delft

HEWLETT PACKARD

Collective Intelligence for Network Control

Ruud Schoonderwoerd

Thesis submitted for the title of 'Ingenieur (ir.)' at Delft University of Technology, Faculty of Technical Mathematics and Informatics



graduation committee:
Prof. dr. H. Koppelaar
Prof. dr. ir. E.J.H. Kerckhoffs
Drs. dr. L.J.M. Rothkrantz
J.L. Bruten M.Sc.