The Implementation and Visualisation of a Large Spatial Individual-Based Model using Fortran 90

Tim Hopkins and David R. Morse
Computing Laboratory, University of Kent at Canterbury,
Canterbury, Kent CT2 7NF, UK

October 10, 1996

Abstract

We look in detail at the implementation of a simulation of the spread of Barley Yellow Dwarf Virus in a barley field. The model considers explicitly each individual plant and aphid, therefore it requires special care to reduce the amount of storage used whilst still producing a computationally efficient code. We attempt to quantify the cost of some of the decisions made in terms of their memory and processor time requirements. Finally we briefly consider the visualisation of the results and how the amount of data produced by the model may be reduced to a manageable level.
KEYWORDS: BARLEY YELLOW DWARF VIRUS, SIMULATION

1 Introduction

The increased power and availability of computers in recent years has led to the development of two new types of ecological simulation model: individual-based models and spatially-explicit models. In the former, each biological entity in the model is simulated at the level of the individual organism rather than the population level (see DeAngelis & Gross [6], Judson [10] and Kawata & Toquenaga [11] for reviews). Among other things, this allows individual variation to be simulated [8]. In the latter type of model, space is represented explicitly, usually on an extended regular array of patches, with communication (or migration) often being restricted to between adjacent patches rather than global communication with all the patches in the model (see for example, Hassell, Comins & May [7], [4]. These spatial simulation models have been developed in recognition of the important part that spatial heterogeneity has to play in influencing population dynamics.

Spatial and individual-based ecological models use substantial computing resources — both processing power and memory [18], hence there are relatively few examples of individual-based spatially-explicit ecological simulation models in the literature. Those models which have been developed have tended to use special purpose, parallel computers such as transputer networks [15], or the Connection Machine [5], [17]. There have been relatively few examples of individual-based spatially-explicit ecological models which have been developed on general purpose computer hardware.

In this paper we describe the implementation of a reasonably simple model for the spread of Barley Yellow Dwarf Virus (BYDV). It is an economically important pest of cereal crops and wild grasses worldwide [16]. It causes yellowing of the leaves and often results in significant yield losses in cereals. The approach used here is to model the individual plants in a cereal crop and the aphids in an attempt to obtain a better understanding of the population dynamics of the aphid and the mechanisms involved in the spread of the virus. Because of the very large amounts of data involved a naive approach is doomed to failure due to storage considerations. We therefore consider a number of ways in which we can reduce the amount of memory required to store data and discuss some of the implications these may have on the resultant run times. Finally, we consider how the results of the simulation may be visualised and discuss how data storage demands may also be reduced during this phase.

In section 3 we look in detail at the steps involved in the simulation and follow that with a careful consideration of the storage requirements of the model and discuss why we chose Fortran 90 as the implementation language. Section 6 provides some timing and performance measurements obtained from running the resultant Fortran 90 implementation on a medium and a large simulation. This is followed by some examples of the way in which an off-the-shelf visualisation system may be used to provide a clear pictorial view of the results along with a discussion on minimising the amount of results data that this requires. We conclude the paper with an evaluation of the final code and a short discussion of how extensions to the model may affect the efficiency of the code.

2 Barley Yellow Dwarf Virus

Barley Yellow Dwarf Virus infects a wide range of grasses worldwide, including cereals. It is one of the most economically important diseases of grasses [16]. The virus can only be transmitted from one plant to another by aphids such as *Rhopalsiphum padi* (L.) feeding on the phloem of the host plant [16]. BYDV does not reproduce within the aphid.

The disease is spread in two ways by the aphids [13]:

- Primary infection takes place when infective, winged (alate) aphids migrate onto the crop from reservoir populations of the virus elsewhere;
- Secondary infection results from dispersal of the offspring of the migrant aphids. Note that these aphids first have to acquire infection by feeding on an infected plant before they can pass it on to another plant.

Control measures aimed at halting the spread of BYDV are targetted at the secondary infection phase by reducing or eliminating the spread of BYDV by the offspring of the infective immigrant aphids. This is achieved by applying an aphicide spray to control the aphids. Comparatively little is known about the factors which determine the rate at which this secondary spread of the disease advances through the crop. However, a recent simulation model developed by McElhany *et al.* [12] has indicated that factors such as the preference of the aphid vector for diseased or healthy hosts plants can be important in determining the rate of secondary spread. Other models of the spread of BYDV have been more or less successful in predicting the spatial and temporal dynamice of the disease and the factors which influence its spread [13], [14].

3 Details of the Simulation

We attempt to simulate the spread of BYDV in a cereal field by keeping track of the position and state of the individual aphids and by considering their effect on individual barley plants. The simulation consists of a sequence of days during which the following events take place in the order indicated:

1. Immigration of aphids: over a defined period of consecutive days at the beginning of the simulation period a predetermined number of winged aphids are randomly placed on the individual barley plants in the field. This random placement models immigrant winged aphids being blown into the field. Every plant is equally likely to be assigned an immigrant aphid. Each immigrant aphid is assumed to be of the same age. A probability threshold is provided which determines which of the immigrant aphids are infected with BYDV.

2. Based on given daily temperature data, the development and reproductive rates for all the aphids in the simulation are calculated.

3. For each aphid in the field,

   (a) Its age is updated based on the daily development rate calculated in step 2 above. Newly born aphids become wingless morphs and wingless morphs die when their computed 'ages' exceed one.
   Note: no winged aphids develop during the simulation.

   (b) Based on the daily reproductive rate, newly born aphids appear on the same plant as their parents.
   Note: these newly born aphids immediately become part of the population, i.e., their ages are first updated and they may move plants on their day of birth.

   (c) Its position may change; movement occurs when a given probability threshold is exceeded. The current simulation allows a choice of two dispersal models

      i. purely random: the aphid is transported to a random point in the field.
      ii. movement is restricted to a nearest neighbour move with the probabilities as given in Figure 1.

   These probabilities were chosen as a first approximation in modelling the tendency of aphids to move between plants in the same row in a cereal field [16]. This movement preference reflects the fact that inter-plant spacing is closer within rows than it is between rows of plants. The movement preferences chosen contrast with those employed by McElhany *et al.* [12] who constrained movement to be to the four nearest neighbours only (excluding the diagonals) and had higher probabilities on within row movement ($p = 0.45$) and lower probabilities on between row movement ($p = 0.05$) than those indicated in Figure 1.

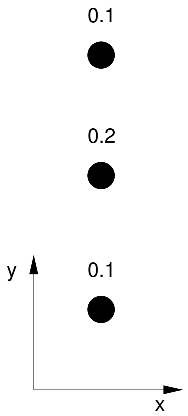

Figure 1: Probabilities of aphid dispersal to the nearest neighbour plant. The rows run parallel to the x-axis.

4. The virus states of the aphid and the plant on which the aphid is feeding are updated. An infected aphid always feeds upon its current plant, passing on the infection. A healthy aphid is always infected if it settles on an infected plant. The incubation periods for both plant and aphid are given and have been assumed to be constant in time.

5. We are primarily interested in visualising

   (a) The spread of the virus among the cereal.

   (b) The population dynamics of the aphid population (the position and the number of infected aphids, and the general population are both of interest).

4 Storage Requirements of the Model

In order to provide a realistic model we require the cereal field to contain between one and ten million plants. This corresponds to a planting density of 300 plants/m^2 and a field plot which is of the order of $100m^2$. We assume that there is an immigrant aphid population of between two and twenty thousand aphids per day during the immigration period. For 10^6 plants we specified an immigrant population of 2000 aphids/day for four days and for 10^7 plants, 20000 aphids/day for four days. The resultant populations after 30 days were 8×10^5 and 8×10^6 aphids respectively. Clearly with this amount of data careful consideration needs to be given to the methods used to represent both the plant and aphid information.
We need to maintain data on the following variables in the simulation model:

- for each individual aphid, its
  - current age (a real value in the range [0,1])
  - life stages (one of newly born, wingless, winged or dead)
  - position in the field (x, y coordinate – a pair of integers)

  - BYDV status (one of infected, incubating, or uninfected)
  - incubation period (the number of days the virus has been incubating – used to update the BYDV status from incubating to infected).

- for each individual plant, its

  - BYDV status (one of infected, incubating, or uninfected)
  - incubation period (the number of days before a plant, bitten by an infected aphid, itself becomes infected – used to update the BYDV status from incubating to infected).

A naive storage scheme might use five integers (one each for the life stage, the x-coordinate, the y-coordinate, the BYDV status and the incubation period) and a real number (the age) for each aphid and two integers for each plant (the BYDV status and the incubation period). (The coordinate information for the plants being implicitly obtained using a two dimensional array.) This gives conservative storage requirements of

10^6 plants $+8 * 10^5$ aphids = 27 Mbytes
10^7 plants $+8 * 10^6$ aphids = 270 Mbytes

where both integers and reals are assumed to require 32 bits. Even with the current cheap state of memory the higher figure is unlikely to be available to most research workers. The use of integer and real arrays would make access to the data relatively fast, although this would necessitate allocating storage space in advance in statically allocated languages such as Fortran 77 with the accompanying problem of accurately estimating an upper bound on the final number of aphids.

Therefore a naive implementation of the model would represent the data as arrays of records (or their equivalent if the programming language does not support structured data types); a one-dimensional array for the aphids and a two-dimensional array for the plants.

4.1 Improvements to the Storage Requirements

We can reduce the storage requirements of the model by noting that the two dimensional view of the data is not necessary for coding the simulation, as all accesses to plants can be directly mapped into a one dimensional array. The actual coordinates of each plant are only required for nearest neighbour calculations and the final visualisation of the data. This reduces the field position of which plant the aphid is on to a single integer.

To avoid the need to reserve array space we implement the storage of the aphid data as a linked list of records, where each individual aphid has its age, life state, position, BYDV status and incubation period recorded. The major disadvantage of a simple linked list over an array is the loss of fast random access to the data. However, in this application, we are only interested in accessing the aphids sequentially. There are also time overheads involved in packing and unpacking the records to extract the required data, and the hidden storage overhead of the pointers involved in the linked list.

The data within the aphid record can be compressed even further, again decreasing the storage required, but further increasing the overheads of packing and unpacking the data. For example, the aphid data could be represented as follows

- age = 32 bits (real)
- life status = 2 bits (only 4 possible states)
- position in the field = 24 bits (range $0 \rightarrow 10^7$ i.e., $\approx 4000 \times 4000$ plants)
- BYDV status = 2 bits (only 3 possible states)
- incubation period = 4 bits (allows up to 15 days).

It should be noted that the last four fields may be packed into a 32 bit integer and thus the data associated with each aphid has been compressed to 8 bytes and a pointer (normally 4 bytes). The storage requirements given earlier in this section are then reduced to 17.6Mb and 176Mb respectively.

We note that although the age of the aphid requires updating at each time step, if no other data associated with that aphid changes, there is no need to repack the compressed data.

We require to access the plant data randomly (typically we wish to ascertain efficiently whether the plant at a given coordinate is infected) so linked lists cannot be considered practical because of their sequential access mechanism. As far as the simulation of the plant population is concerned we are only interested in whether a plant is infected or not. Thus provided we temporarily store the positions of the incubating plants in some efficient manner we may reduce the plant array to a bit array. Provided the incubation period is short (4 days is typical), and that not too many plants are incubating at any time, we may also store the incubating plants as linked lists. This has a storage overhead of an integer and one pointer for each incubating plant.

The final structure used for storing the plant data was a bit array to store the infected/uninfected state of each plant along with a circular list of linked lists storing the coordinates of incubating plants. On completing the incubation period these plant lists are used to update the bit array and the storage is reused. Figure 2 gives a pictorial view of this data structure.

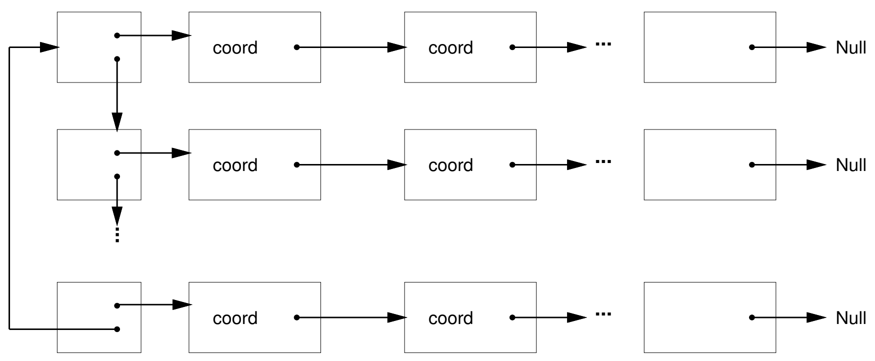

Figure 2: Incubating plants data structure (see text for further explanations).

The circular list at the head of the data structure just provides easy updating of the pointers recording the current day and end of incubation day pointers.

The storage space required is then reduced to $10^6/8 + 8n_1$ bytes for 10^6 plants and $10^7/8 + 8n_2$ bytes for 10^7 plants where n_1 and n_2 are the maximum number of plants incubating at any one time.

An alternative method of storing the plant data would be to allocate an extra incubation bit for each plant. This method is only more efficient if $> 1.5\%$ of all the plants are infected over any incubation period.

5 Language Choice

Any language supporting the use of pointers and providing facilities for the simple manipulation of bits would be suitable for implementing the simulation; two of the more popular languags are C and Fortran 90. Fortran 90 [9] was chosen since it provides the ability, via HPF (High Performance Fortran [1], [2], [3]), to port the code onto a parallel architecture such as a DEC Alpha Cluster. In addition, by making use of the module facility available in the new Fortran language, we may use information hiding. This means that it is possible to build the simulation software so that the user is unaware of the underpinning data structures. Indeed the effects of any change to the data structure are localised within a single module and, since all access to this module is at a subroutine level, no changes are required elsewhere in the simulation code.

For example, the aphid-control module contains a number of publically callable routines which allow

1. an aphid's record to be unpacked into its components and repacked into a, possibly compressed, record;
2. newly born aphids to be added to the list and dead ones deleted;
3. a count to be kept of the number of aphids at each life stage, and so on.

A similar approach may be used for manipulating the plant data, both for storing the incubating plants and for recording those infected.

6 Performance of the Simulation Software

All code was developed on a Silicon Graphics Indy with a 133MHz R4000 processor and 32Mbytes of memory running Irix 5.3 and using the Edinburgh Portable Compiler Fortan 90 system. It was also successfully compiled and run on a Sun Sparc 10 using the Craysoft and NAG Fortran 90 systems without any source code changes.

The results reported in this section refer to simulations on square grids of p plants in each direction (i.e., p^2 plants in total) with $0.002p^2$ immigrant aphids on each of the first four days. The immigrant aphids are all winged aphids aged 0.5 with a probability of 0.1 of being infected with BYDV. The probability that an aphid moves during the course of a day is 0.05. The simulation takes place between days 70 and 100.

All timings were obtained using the Fortran 90 intrinsic function *system_clock*. Care was taken to ensure that the machine was, as far as is possible with any networked machine, only processing the simulation model. Multiple runs were made and the shortest reported time used.

Table 3 shows the total CPU time used along with the final number of aphids for $p = 500$, 1000 and 2000 using a linked list to store the aphid information and packing the coordinate, life stage, incubation time and BYDV status into a single integer.

# Plants	# Immigrant Aphids	Final # Aphids	CPU seconds
2.5×10^5	2000	2×10^5	8.5
10^6	8000	8×10^5	41
2.5×10^6	20000	1.8×10^6	130
4×10^6	32000	3.2×10^6	460

Figure 3: Results on 133MHz SG Indy using EPC Fortran 90 with highest level of optimization

A more detailed breakdown of these timings is given in Figure 4 where the CPU time required for each day of the simulation is plotted. It may clearly be seen that, as p increases, the CPU time required to process each day of the simulation increases far more rapidly towards the end of simulation. Indeed the plot for $p = 1000$ is almost linear over the last ten days whilst it shows a very steep rise for larger values of p. The reason for the large jumps at day 99 for $p = 1500$ and at day 96 for $p = 2000$ may be explained by the dramatic increase in the number of page faults (from 1 when $p = 1000$ to 96,800 when $p = 2000$) which take place as the data structures grow. It may also be the case that the run time system is attempting to free extra space by garbage collecting. In real terms, for $p = 2000$, this book-keeping work by the system is accounting for around 50% of the processor time used for the simulation.

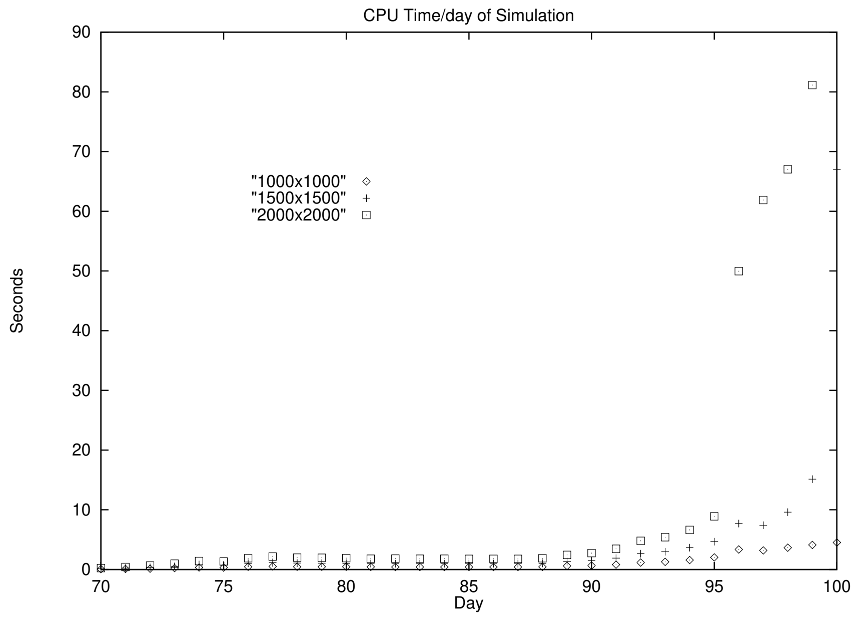

Figure 4: CPU time required for each day of the simulation

Figure 5 shows the number of records in the linked list storing the aphid data at the end of each day, this includes all newly born, wingless and winged aphids. Any aphids which die or go outside the grid are deleted from the list. The flat portion of the curve (days 78–88) shows the period between the immigrant aphids dying and the newly born aphids reaching reproductive age.

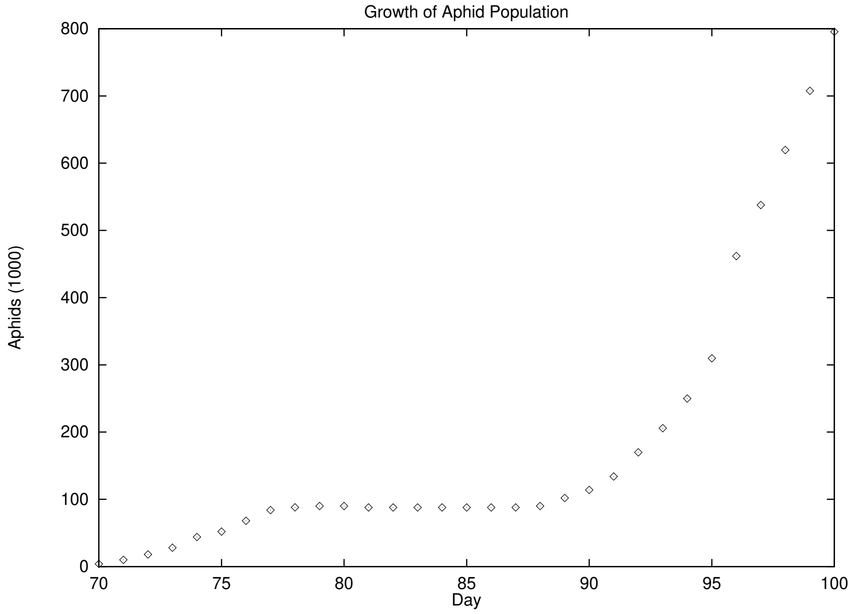

Figure 5: Aphid population for each day of the simulation ($p = 1000$)

Figure 6 shows the growth in the total number of plants becoming infected with time. The bottom line shows the number of plants becoming infected on each day of the simulation and the middle line shows how many plants are being stored in the data structure for that day. This is one area in which the data storage could be improved. The problem is that if a plant is bitten by more than one aphid multiple copies of its coordinates are stored in the linked list. As the aphid population grows this problem becomes more pronounced, the excess is, however, quite small when compared to the storage required for the extra bit for each plant necessary to efficiently prevent duplicate entries.

Packing the aphid records into an integer and a real as described in Section 4.1 did allow larger problems to be tackled. Tests using this extra packing indicated an overhead of approximately 15%.

Finally the simulation was implemented using an array of packed records; this provided the saving in space but required setting the length of the array storing the aphid data at the start of the simulation. The execution times are comparable for the smaller values of p but for the larger domains ($p = 1500$ and $p = 2000$) they do not exhibit the page fault problem to the same degree (just 10,000 page faults for $p = 2000$). The run time for $p = 2000$ was 200 seconds with the effects of swapping not evident until day 97. The delayed effects of swapping are due to the fact that using arrays does not incur storage overheads for pointers.

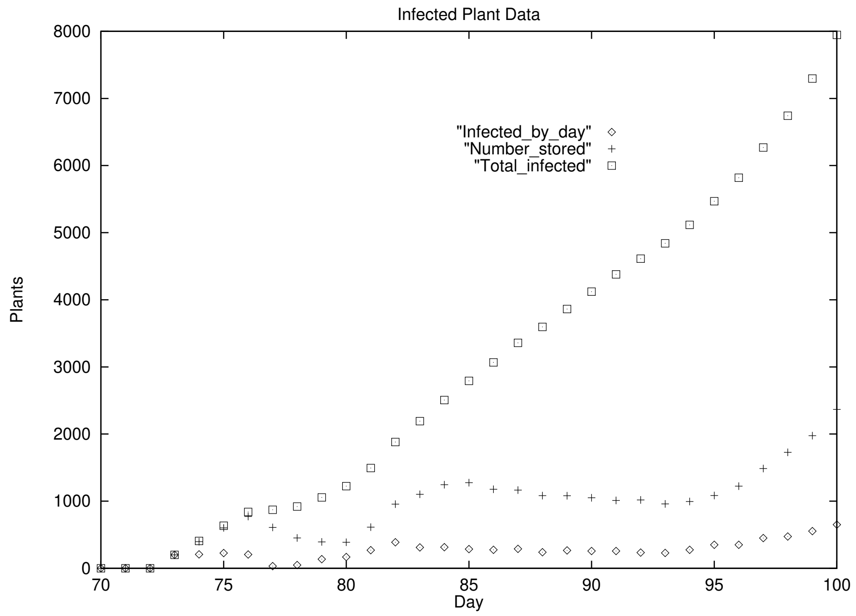

Figure 6: Aphid population for each day of the simulation

7 Visualizing the output

The simulation contains state information on at least one million plants and one million aphids, each of which has the potential to be updated during each time step (equivalent to a day) of the simulation. While this data could be summarized in one or a few values, (see for example Figures 5 and 6), clearly much information is lost, particularly on the spatial aspects of the dynamics of the model. Some data reduction is almost inevitable given the amount of storage which would be required to store all the data generated by the simulation and its dimensionality. There are also too many variables in the data for them to be simultaneously displayed and interpreted easily.

One of the most basic views of the output of the simulation is an animation of the spread of infected plants and aphids as the simulation progresses. Overlaid on top of this could be contour maps or iso-surfaces showing the distribution and abundance of the aphids — both the infected and the uninfected populations. A single frame from such an animation (corresponding to the state of the simulation at day 100) is shown in Figure 7.

There are at least two ways in which the data required for such a display can be produced. One is to dump the state of the entire simulation at the end of each time step; the second is just to produce the changes in the state of the simulation which have occurred during the last time step. There is a trade-off here which depends on what proportion of the simulation state changes. With simulations such as the BYDV simulation where the stored state is large compared to the changes between one time step and the next (particularly in the early stages of the simulation when the size of the infected patches are small) then the former approach is preferable. For example, in most simulations, the size of the file containing all the changes to a simple map of the spread of infected plants for an entire simulation is smaller than that which would have been produced by dumping the state of infection of all the plants in the simulation in a single time step. In other simulations such as those involving cellular automata or those of Hassell, Comins & May [7], [4] where virtually all the simulation state changes at each time step, then the saving through just producing the changes in state are minimal or nonexistent.

Other ways in which the size of the data files which the simulation produced could be reduced were to use standard UNIX file compression tools and to write the files as binary files rather than ASCII files.

As well as the data storage issue, a second issue is extracting the required state information from the simulation. In general, state information which was stored in arrays, such as the plant infection status could be extracted easily simply by printing out the arrays at each time step. Other information was more difficult to extract, such as the aphid data (which was stored in linked lists) although summary information could be accumulated at each time step as each aphid was processed. Relating the aphid and plant states and the changes in state and movement of the aphids was particularly difficult and involved generating extremely large data files which were then post-processed by simple programs to extract the relevant data in a form which could be visualized.

In general it proved simpler to separate physically the simulation and visualization functions of the modelling process by using graphical and visualization packages such as Uniras and Explorer. These were linked to the simulation by using temporary files. An alternative approach would have been to have included bespoke graphical display facilities in the simulation. While this would have made the simulation easier to use, it would have reduced the potential for analysing the simulation output in non-standard ways.

8 Summary and Conclusions

We have used Fortran 90 to produce an efficient implementation of a relatively simple individual-based model for the spread of BYDV within a cereal field. The new pointer facilities available in Fortran 90 allowed us to use data structures that grew with the aphid population and the bit level intrinsic functions provided a portable, storage efficient means of storing the state (infected/uninfected) of each plant. These same intrinsic functions also meant that we could compress the data describing each individual aphid as far as possible and this allowed much larger problems to be solved, albeit with the computational overhead of packing and unpacking the data.

The use of pointers does lead to overheads in both storage and CPU time due to page faults as the data structures grow. The use of arrays, which required length information to be provided at compile time, alleviated this problem by reducing the number of page faults for a given size problem. Certainly the move to High Performance Fortran (HPF) will necessitate the use of an array based data structure although individual aphid data could still be packed. The advantage which HPF brings is that it makes it much easier to distribute the simulation over multiple processors, which is another way of improving the performance of such simulations.

One of the problems with individual-based models is that of visualizing the vast amounts of data that describe the state of the model at each stage of the simulation. With our relatively simple model we were primarily concerned with the spread of the disease within the plants. We could thus restrict the data required to update our view of the plants at the end of each time step to just the coordinates of any newly infected plants. To obtain a detailed view of the aphid population is more complex. Some information like the number of aphids at each life stage or the number of infected and uninfected aphids can easily be updated as the aphid data is sequentially processed. Other views of the data, especially those requiring links between the plants and the aphids were more difficult to obtain and generated very large data files. Improvements may be possible in this area by recording aphid movements and state changes and post-processing these to form animations.

Future work in this area will investigate both distributing the simulation among a number of processors using HPF and increasing the complexity and biological realism of the actual model. We will also be considering how we can couple the computation and visualization parts of the simulation more tightly together. Such a coupling will allow a more interactive exploration of the behaviour of the simulation under various parameter combinations and changes rather than a post-hoc analysis of the results. We are also investigating how simulation and visualization tools can be linked in a network transparent manner so that the two essential components of the modelling process can be run in parallel on different processors.

References

[1] Anonymous. High Performance Fortran language specification (part I). *ACM Fortran Forum*, 12(4):1–86, December 1993.

[2] Anonymous. High Performance Fortran language specification (part II). *ACM Fortran Forum*, 13(2):87–150, June 1994.

[3] Anonymous. High Performance Fortran language specification (part III). *ACM Fortran Forum*, 13(3):22–55, September 1994.

[4] H.N. Comins, M.P. Hassell, and R.M. May. The spatial dynamics of host parasitoid systems. *Journal of Animal Ecology*, 61(3):735–748, 1992.

[5] R. Costanza and T. Maxwell. Spatial ecosystem modeling using parallel processors. *Ecological Modelling*, 58(1–4):159–183, 1991.

[6] D.L. DeAngelis and L.J. Gross. *Individual-based models and approaches in ecology*. Chapman & Hall, London, 1992.

[7] M.P. Hassell, H.N. Comins, and R.M. May. Spatial structure and chaos in insect population-dynamics. *Nature*, 353(6341):255–258, 1991.

[8] M. Huston, D. Deangelis, and W. Post. New computer-models unify ecological theory. *Bioscience*, 38(10):682–691, 1988.

[9] ISO/IEC. *Information Technology – Programming Languages – Fortran (ISO/IEC 1539:1991(E))*. ISO/IEC Copyright Office, Geneva, 1991.

[10] O.P. Judson. The rise of the individual-based model in ecology. *Trends in Ecology & Evolution*, 9(1):9–14, 1994.

[11] M. Kawata and Y. Toquenaga. From artificial individuals to global patterns. *Trends in Ecology & Evolution*, 9(11):417–421, 1994.

[12] P. McElhany, L.A. Real, and A.G. Power. Vector preference and disease dynamics — a study of barley yellow dwarf virus. *Ecology*, 76(2):444–457, 1995.

[13] D. Morgan. A simulation model of BYDV epidemiology. *Proceedings of CYMMIT Workshop on Barley Yellow Dwarf Virus — 1987*, pages 300–304, 1989.

[14] D. Morgan, N. Carter, and P.C. Jepson. Modelling principles in relation to the epidemiology of barley yellow dwarf virus. *Bulletin IOBC/WPRS*, 11:27–32, 1988.

[15] D.R. Morse. Spatial simulation modelling of insect population dynamics on a transputer network. In J. Kerridge, editor, *Transputers and occam research: new directions*, pages 66–75, Netherlands, 1993. IOS Press.

[16] A.G. Power. Competition between viruses in a complex plant-pathogen system. *Ecology*, 77(4):1004–1010, 1996.

[17] E. Uziel and M.W. Berry. Parallel models of animal migration in Northern Yellowstone National Park. *International Journal of Supercomputer Applications and High Performance Computing*, 9(4):237–255, 1995.

[18] F. Villa. New computer architectures as tools for ecological thought. *Trends in Ecology & Evolution*, 7(6):179–183, 1992.

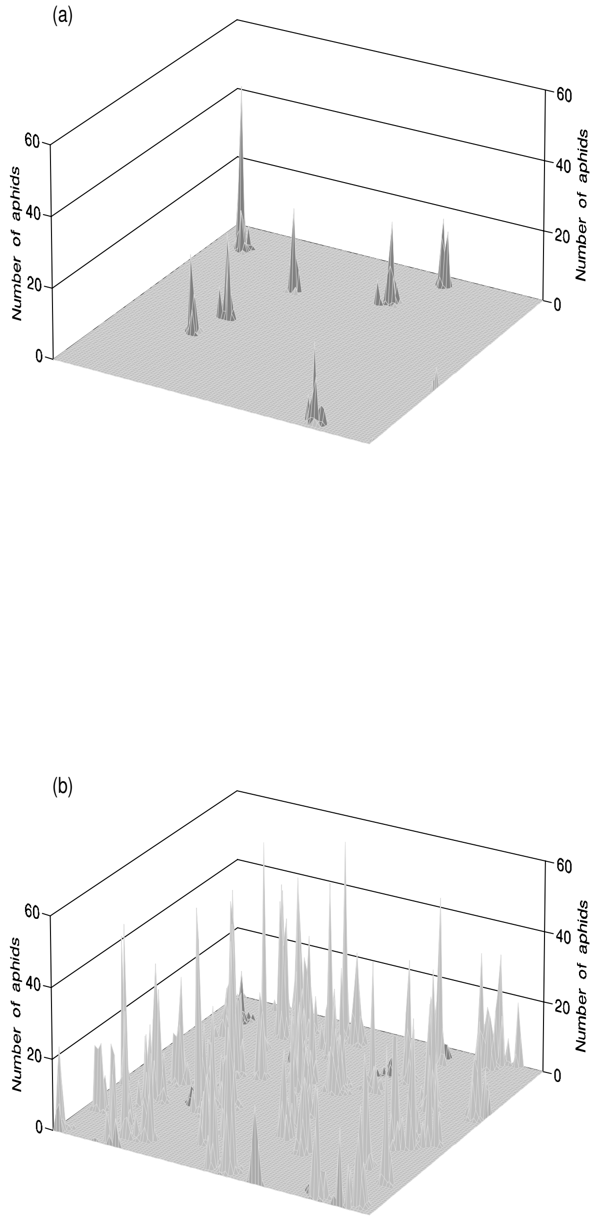

Figure 7: (a) Healthy and (b) infected aphid numbers on each plant in a simulation on a 100×100 lattice. The dark grey areas denote plants which have been infected by the BYDV virus. (The size of the simulation was reduced for clarity on a small figure.)