Tools and Techniques for Optimisation of Microscopic Artificial Life Simulation Models

C. J. Scogings and K. A. Hawick and H. A. James

2006

We have developed a sophisticated microscopic model for studying emergent spatial patterns in a predator-prey based artificial life scenario. In this paper we describe some of the simulation optimisation tools and techniques that are necessary to support many long model runs over different parameters and starting conditions. We discuss how these optimisation techniques may be of general value to developers of other simulation codes.

Keywords: simulation; optimisation; simulation tools; simulation visualisation

BiBTeX reference:

```
@INPROCEEDINGS{CSTN-033,
        author = {C. J. Scogings and K. A. Hawick and H. A. James},
        title = {Tools and Techniques for Optimisation of Microscopic Artificial Life
                Simulation Models},
        booktitle = {Proceedings of the Sixth IASTED International Conference on Modelling,
                Simulation, and Optimization},
        year = {2006},
        editor = {H. Nyongesa},
        pages = {90-95},
        address = {Gabarone, Botswana},
        month = {11-13 September},
        publisher = {IASTED},
        keywords = {simulation; optimisation; simulation tools; simulation visualisation}
}
```

Technical Report CSTN-033

Tools and Techniques for Optimisation of Microscopic Artificial Life Simulation Models

C.J. Scogings, K.A. Hawick and H.A. James
Institute of Information & Mathematical Sciences
Massey University – Albany
North Shore 102-904, Auckland, New Zealand
email: {c.scogings, k.a.hawick, h.a.james}@massey.ac.nz

7 April 2006

ABSTRACT

We have developed a sophisticated microscopic model for studying emergent spatial patterns in a predator-prey based artificial life scenario. In this paper we describe some of the simulation optimisation tools and techniques that are necessary to support many long model runs over different parameters and starting conditions. We discuss how these optimisation techniques may be of general value to developers of other simulation codes.

KEY WORDS
simulation; optimisation; simulation tools; simulation visualisation.

1 Introduction

We have developed a predator prey model for studying emergent behaviours [2] in artificial life models. Our model has been previously described in [8] and [7] and the emerging clusters formed by its "artificial animals" (animats [12]) have been described in [3] and [4]. The differences and similarities between our model and other well-known models such as Tierra [13] and Avida [1] have been outlined in [5]. In brief our model consists of a "flatland" in which animal agents or animats live, die, breed and interact with one another. A key feature of our model is its use of two broadly different classes of animat - predator and prey species. A unique feature of our model is its use of prioritised rules that determine individual animat behaviours.

In order to produce interesting emergent behaviours it is important for the model to be run for a significantly large number of time steps and for it to contain a significant number of animats – usually a minimum of 5000 to reproduce the phenomena we discuss. At each time step, every animat needs to change its state based on the locations and state of its neighbours. It is this process of finding the nearest neighbours that dramatically increases the time required to perform a useful run of the model.

One way to increase execution speed is to run the model in parallel and this has been reported in [9]. However a parallel supercomputer is not always available and even in a parallel implementation it is useful to increase sequential efficiency as much as possible. One way of saving time was to remove the need for using the square root function when calculating distances and this is reported in [6].

The greatest gains in efficiency were achieved by modifying the way that animat agents are represented within their environment. Thus a succession of techniques was introduced into the model to reduce the time taken for an animat to search for its neighbours.

In section 2 we summarise the core features of our animat model. The main contribution of this present paper is a discussion and performance comparison of the four main algorithmic ideas we employed to speed up model simulations. We describe these in section 3 and give some detailed performance results in section 4. In section 5 we discuss the performance gained from the algorithms and conjecture how they might be applied to other spatial simulation model codes.

2 The Core Model

We developed out model as a platform to experiment with the large scale emergent patterns that occur in nature and indeed in many models. We are interested in studying herding, pack hunting and other large scale behaviour patterns. These are not directly incorporated into the model however but only emerge as a consequence of the interplay between detailed microscopic parameters that we only specify for individual animats.

Our core model is a predator-prey model containing two species of animat - the predators (foxes) and prey (rabbits). Rabbits are considered to have an unlimited amount of food available while foxes have to eat rabbits to survive. In theory animats are free to move on an infinite plain and this freedom of movement and any restrictions thereon are considered in the following sections. In practice, we terminate model runs when the number of animats in the program exceeds available computer memory. Animats have a life span and may die of old age or may be eaten (prey only).

Each animat contains a set of rules. For the work reported in this paper these rules are fixed and no evolution is taking place. [1] Thus any new animat is an exact clone of its parents. The rules are consulted in the order given below. The first applicable rule in the list is executed and then all other rules are ignored. Thus every animat executes exactly one rule every time step.

The rules for prey (rabbits) are as follows:

1. move away from a fox if the fox is adjacent
2. breed if another rabbit is adjacent and less than 5 rabbits are nearby
3. move towards a rabbit if that rabbit is within vision range
4. move randomly to an adjacent position

We have experimented with specific parameter values such as the overcrowding parameter "5 rabbits" in rule 2 above and the "vision horizon" parameter referred to in rule 3 above. Generally the model is somewhat insensitive to changes in these. We discuss suitable values elsewhere [3].

The rules for predators (foxes) are as follows:

[1]It is possible to allow animals to adapt their rule priorities. We are presently working on this as a form of evolved behaviour.

1. eat a rabbit if the rabbit is adjacent
2. move towards a rabbit if the rabbit is within vision range and the fox is hungry
3. breed if another fox is adjacent and less than 3 foxes are nearby
4. move towards a fox if that fox is within vision range and this fox is not hungry
5. move randomly to an adjacent position

The breeding process consists of checking whether a random number is less than the current birth rate. If this is the case a new animat is produced at the same location as the parent. Birth rate can be evaluated globally or locally according to a local density of animals function that mimics food supply availability for rabbits.

3 Optimisations Tools and Techniques

In this section we describe four techniques or methods for time-stepping the animat agents in the simulation program. We show how *a priori* known properties of the animats' neighbours and finite limits of visionary range can be exploited to rearrange the update sweep and optimise the performance of our simulation code.

Two important underpinning issues are those of identifying the animat's nearest neighbours and choosing an appropriate vision range or horizon to determine what "neighbouring" means in the model context.

3.1 Nearest Neighbours

It is immediately obvious from the animat rules (above) that every animat needs to know the location of its nearest neighbour. In fact it needs to know the location of its nearest neighbour of its **own species** in order to breed with it and it also needs to know its nearest neighbour of the opposite species in order to eat it or to flee from it. Thus at every time step of the model, every animat needs to locate two different nearest neighbours. Previous work has found that it is this calculation of the nearest neighbours that consumes most of the processing time of the model.

3.2 Animat Vision Range

Every animat has a vision range which is the distance within which it will react to other animats. Thus the model rule “move towards the nearest animat of the same species” will only be applied if the nearest neighbour is within the vision radius. If an animat becomes isolated and all other animats are outside its vision range it will not record any nearest neighbours and will not apply any of the rules involving neighbours. In fact the only rule an isolated animat can apply is “move randomly” in the hope of moving within vision range of another animat.

Thus the nearest neighbour problem, as far as the predator-prey model is concerned, can be restated as - find the nearest neighbour within the vision range and ignore any animats that are not within vision range.

We now discuss each of our four methods, listed as A,B,C,D in order of performance improvement.

3.3 Method A : The Infinite Plain

When the predator-prey model was initially constructed, the animats were placed in an infinite plain. This was important as previous Artificial Life [11] models were always situated in a confined environment (see [7, 8]). The infinite plain enabled observations of emergent behaviour (see [6, 9]) without the constraints imposed by a bounded area.

However, the “infinite plain” approach contained a significant disadvantage in that the locations of animats were not grouped within any sort of grid system. In fact this was implemented with a simple **list of animats** data structure, with no spatial information directly indexable. This meant that the only way animats could search for nearest neighbours was to perform an exhaustive search on **all** the other animats and this slowed the model considerably.

In pseudo-code, this algorithm is:

```
for each animat A in the list
  for each other animat B in the list
    is B a possible neighbour of A ?
  end-for
  update A based on chosen B neighbours
end-for
```

It was necessary to always search through the list of all other animats because at each time step, any other animat could move into the vision range or out of the vision range so there is no way of predicting the nearest neighbours. In addition animats can disappear due to “death” by old age or by being eaten.

3.4 Method B : The Expandable Grid

An “expandable grid” was placed over the infinite plain. Each square of the grid covered several locations and maintained a list of all animats situated within that grid square. Whenever an animat left the area covered by the grid, a new grid square would be created to cover the area that the animat had moved into. The size of each grid square was slightly larger than the vision range of an animat. Thus an animat positioned near the centre of a grid square would only have to search for nearest neighbours within its own grid square and there would be no need to perform the exhaustive search through the full list of animats. This is effectively a spatial hashing algorithm.

In pseudo-code, this algorithm is:

```
for each animat A in the list
  for each other animat B
           that is in A's grid square
    is B a possible neighbour of A ?
  end-for
  update A based on chosen B neighbours
end-for
```

This system reduced the time required to locate the nearest neighbours even though many animats had to search through two or more grid squares due to being positioned near to the edge of a particular grid square.

A disadvantage of this system however, was that animats tended to cluster (see [3, 5]) and thus a large number of animats could still be found within one grid square, or across adjacent grid squares, leading to lengthy searches for nearest neighbours.

3.5 Method C : The Matrix

Meticulous analyses of the emergent behaviour of animats (see [3, 4]) showed that an infinite plain was not in fact required and that although the model could remain spatially open bounded, in practice a sufficiently large matrix array could contain all animats within a finite simulation run. This was because animats tended to cluster together in order to breed and did not move significant distances from the point of origin. It was therefore decided to place a fine-grained grid over the area in which animat activity occurred. This grid would contain only one location in each grid square (unlike the expandable grid in Method B above) and could thus be represented as a two-dimensional matrix array of locations. Animats are able to multi-occupy a single spatial location so each matrix entry still required a possible list of animats within that location. In effect the key data structure here is a two-dimensional array or matrix where each cell is an open ended list of occupying animats.

The search for nearest neighbours was now able to take place solely within the vision range of the animat by checking only those matrix locations which are within the vision range. This significantly improved the execution speed of the model. A pseudo-code outline of this check is provided below.

```
for x = ax ; v to ax + v do
  for y = ay ; v to ay + v do
    look for neighbour at matrix[x, y]
  end-for
end-for
```

Note that the animat is located at (ax, ay) and the vision range is v.

This approach can encounter problems when the animat is situated close to a boundary of the matrix and the check then attempts to reference a matrix element which is out of bounds. This problem is solved by stopping animats from moving to within vision range of the edge of the matrix. This is illustrated in figure 1.

Another problem with this approach is that the area being searched is a square around the animat whereas it should actually be a circle. Thus certain locations are included in this search (notably along the diagonals) when they are actually outside the vision range of the animat. This means that the distance has to be calculated from the animat to each location in order to ensure that the location is actually within the vision range. These distance calculations are a source of speed reduction and are addressed in Method D below.

The disadvantage of the matrix approach is that the animats now exist in a bounded plain and it is possible that new emergent behaviours may be affected by the boundary constraints. This problem will need to be monitored in future work perhaps a log could be maintained of how many animats reach the edge of the matrix. Of course if the matrix can be made large enough this problem will diminish.

3.6 Method D : Table-Lookup

This method retains the matrix of Method C but speeds up the search process through locations within the vision range. It was recognised that a **precomputed** list of spatial index offsets could be created that allowed the rapid calculation of which matrix locations needed to be checked during the search for nearest neighbours.

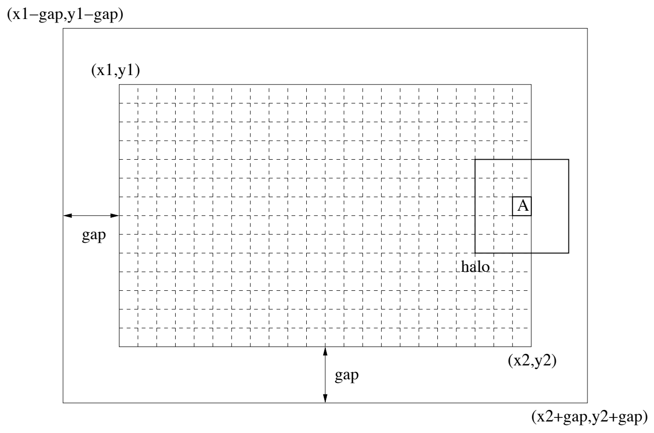

Figure 1: The matrix is constructed to be larger than the area in which the animats can exist (x:x2, y1:y2). This creates a "gap" around the edge of the structure which must be greater than the animat vision range, thereby increasing the efficiency of searching for neighbours because checks for out-of-bound conditions are not required. This algorithm pertains to Methods C and D. Animat "A' can therefore safely probe the matrix contents around its halo without ever experiencing array-out-of-bounds issues. Gap elements of the matrix simply store a "not used" value.

For example if the vision range is 3 then the following (x, y) offsets are pre-calculated and they are stored in this order, i.e. in order of radial distance from the origin.

```
( 0,  0)
(-1,  0) ( 0, -1) ( 0,  1) ( 1, 0)
(-1, -1) (-1,  1) ( 1, -1) ( 1, 1)
(-2,  0) ( 0, -2) ( 0,  2) ( 2, 0)
(-2, -1) (-2,  1) (-1, -2) (-1, 2)
( 1, -2) ( 1,  2) ( 2, -1) ( 2, 1)
(-2, -2) (-2,  2) ( 2, -2) ( 2, 2)
(-3,  0) ( 0, -3) ( 0,  3) ( 3, 0)
```

This list can be additionally sorted (by secondary sort key) in angular order to support a deterministic spiral search for a neighbour, or within each radial slice the cells can be randomly shuffled by angle to avoid introducing any spatial structural bias.

Figure 2 illustrates a small example of a list of cells at a fixed radius. Note that the discretisation of the mesh coordinates does mean there is an aliasing effect introduced at small radii. As far as we can determine the model is not sensitive to this effect however.

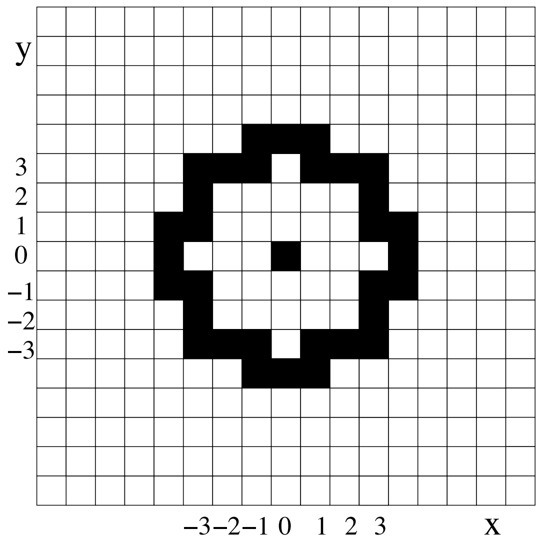

Figure 2: The radial lookup table prescribes those spatial cells that might contain another animal. The cells can be stored in a lookup table that is sorted by radius so that only the minimal list of possible positions necessary to find the "nearest neighbour" will be traversed.

Since the index and associated spatial radius calculations are no longer in the inner loop of the code, some significant speedup accrues. The method then follows two basic steps: an initialisation step and then for each animat, at every time step, the following pseudo-code is executed.

Initialisation: A list of (x, y) offsets is calculated and stored in a list. The offsets are sorted into order of the distance of their respective locations from the origin. Note that this is an initialisation step and occurs once only prior to the model getting underway. The calculation of the offset list uses the vision range of the animats. If different types of animat have different vision ranges (for example predators usually have a greater vision range than prey) then different offset lists can be created, one list for each species.

For every animat (located at (ax, ay)) at each time step during the execution of the model:

```
while (data in the offset list) do:
  get x-offset and y-offset from the list
  x = ax + x-offset
  y = ay + y-offset
  look for neighbour at matrix[x, y]
  if (nearest neighbours found) then:
     exit-loop
  end-if
end-while
```

This approach has two advantages. Firstly no distance calculations are required as each animat only checks locations (calculated from the list of offsets) that are definitely within the vision range. The second advantage is that the offsets are provided sorted in order of distance. Thus as soon as an animat discovers a neighbour nearby the search through other locations can be terminated early as the newly discovered neighbour must be the nearest one. So any animat that has discovered one neighbour of each species can have its search terminated.

4 Performance Results

Figure 3 shows the improvements in run time for typical runs using the four algorithms described.

As can be seen, performance improvement is monotonic over the range of animals in the system relevant to the work reported in this paper.

Method A (Infinite Plain) shows the classic order $O(n^2)$ behaviour for n animals where all n animals interact with every other animal.

Method B, using a Grid, shows a drastic improvement in performance time, however it is no longer monotonically trivial since there is of necessity a set of cross-over points where the number of animals in a grid element spills over into the next element, thus reducing the number of neighbours to search in each grid element. This manifests itself in the peaks in the total time at, for example time-step 5000.

Method C shows even further improvements over Methods A and B because this approach maintains a "matrix" of spatial cells so that neighbour calculations can be done more rapidly via direct spatial addressing rather than scan a list of arbitrarily-located animals (many of which will not be neighbours). However, Method C still requires distance calculations as it employs a square-shaped halo region around the animat in question.

This effect is completely removed in Method D, where the matrix is retained but we employ a lookup table of pre-calculated off-sets. Furthermore by pre-sorting the offset table by radius we enable the animat to optimally search for the closest neighbouring animat to it. Therefore, as figure 3 shows, algorithm D has consistently the best performance out of those studies.

We believe these results show the effectiveness of

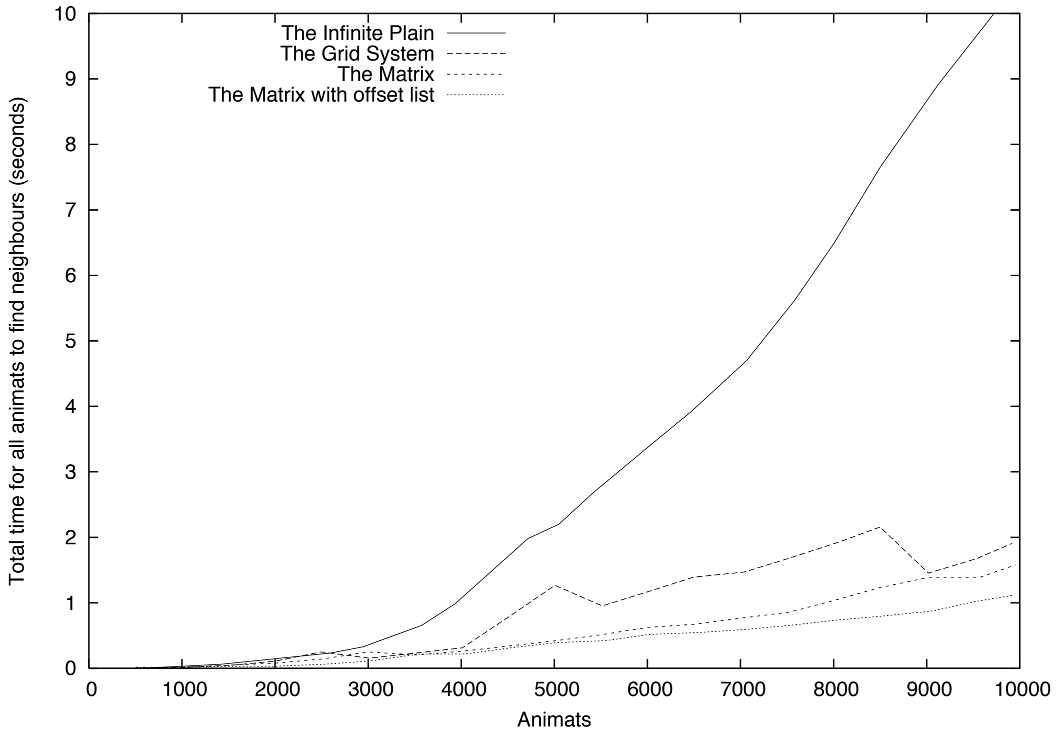

Figure 3: Run time of the simulation program using the four different spatial algorithmic approaches: "Infinite Plain (A); Grid System (B); Matrix (C) and Matrix with Offset List (D). The algorithms show continuous improvement although "the Matrix" algorithm (Method C) shows execution time peaks around 2500, 5000 and 6500 animats where animals "spill over" into nearby containment grids. This effect is less significant when we average performance measurements over several independent runs and which generate microscopically different (albeit statistically similar) patterns.

these approaches and that they may have some applicability in other spatially-oriented programmes.

5 Summary and Conclusions

We can determine the approximate complexity of each algorithm by plotting the timing data shown in figure 3 on a log-log scale. Hypothesising that $t \simeq O(n^p)$ for time t and n animats, a straight line fit or linear regression fit to the log-log data yields the dominant power p as the gradient of the fit since we obtain $\log(t) \simeq p\log(n) + \text{const.}$

Table 1 shows approximate values for the dominant power in the measured complexity of each algorithm. These are of course only approximate given that the measured values smear out all other scaling effects in the code. Nevertheless it is clear that there is a quantitative and monotonic improvement in our series of algorithms and that algorithm method D in particular scales with only around $t \simeq n^{1.5}$ which does allow us to explore significantly large animats numbers to significantly long simulation times.

Algorithm (method)	Fitted Gradient $= p$
A	2.69 ± 0.05
B	2.10 ± 0.1
C	1.73 ± 0.06
D	1.50 ± 0.06

Table 1: Fitted gradients to $log(t)$ vs $\log(n)$ indicating upper bounds on the limiting dominant power in the complexity of each algorithm

In developing simulation codes many developers like ourselves will require to build custom codes rather than using off-the-shelf simulation packages. This is inevitably necessary if we wish to explore multi-scale [10] behaviours for long times and large system sizes.

The algorithms we have described here would still be applicable to a three-dimensional model since the radial lookup index values can just as easily be precomputed for three dimensions as for two. Also even if we were to add spatial topographical data to our "flat world" (to make the spatial geography less uniform) the approaches used in Methods C and D are also still applicable. Therefore we believe that the spatial optimisation algorithms we report in this paper will be of general and lasting use to developers of other simulation models than just our predator-prey artificial life system.

Acknowledgements

The authors gratefully acknowledge the contribution of Helix supercomputer time by Massey University and the Allan Wilson Centre in the production of the data reported in this paper.

References

[1] Christoph Adami. Introduction to Artificial Life. Springer-Verlag, 1998. ISBN 0-387-94646-2.

[2] Peter Cariani. Emergence and Artificial Life. In J.D.Farmer C.G.Langton, C.Taylor and S.Rasmussen, editors, Artificial Life II, SFI Studies in the Sciences of Complexity, pp 775–797. Addison Wesley, 1991. ISBN 0-201-52571-2.

[3] K.A. Hawick, C.J. Scogings and H.A. James. "Defensive Spiral Emergence in a Predator-Prey Model," in Proc. Complexity 2004, Cairns, Australia, Dec. 2004, pp 662–674, Editors: Russel Stonier and Qinglong Han and Wei Li.

[4] K.A. Hawick, H.A. James and C.J. Scogings. "Manual and Semi-Automated Classification in a Microscopic Artificial Life Model," in Proc IASTED International Conference on Computational Intelligence (CI 2005), July 2005, Calgary, Canada.

[5] K.A. Hawick, H.A. James and C.J. Scogings. "Roles of Rule-Priority Evolution in Animat Models," in Proc Second Australian Conference on Artificial Life (ACAL 2005), Sydney, Australia, 2005.

[6] K.A. Hawick, H.A. James and C.J. Scogings. "High-performance Spatial Simulations and Optimisations on 64-bit Architectures," to appear Proc Modelling, Simulation and Visualization 2006, Las Vegas, June 2006.

[7] K.A. Hawick, H.A. James and C.J. Scogings. "Grid-Boxing for Spatial Simulation Performance Optimisation," in Proc 39th Simulation Symposium, Huntsville, Alabama, USA, April 2006.

[8] H.A. James, C. Scogings and K.A. Hawick. "A Framework and Simulation Engine for Studying Artificial Life," Research Letters in the Information and Mathematical Sciences ISSN 1175-2777, Information and Mathematical Sciences, Massey University, Albany, North Shore 102-904, Auckland, New Zealand, May 2004. CSTN-007.

[9] H.A. James, C.J. Scogings and K.A. Hawick. "Parallel Synchronisation Issues in Simulating Artificial Life," in Proc. 16th IASTED Int. Conf. on Parallel and Distributed Computing and Systems (PDCS), pp, 815–820, Boston, November 2004.

[10] Leo P. Kadanoff. *Statistical Physics Statics, Dynamics and Renormalization.* World Scientific ISBN 981-02-3764-2

[11] S. Levy, *Artificial Life The Quest for a New Creation.* Penguin. ISBN 0-14-023105-6, 1992.

[12] Jean-Arcady Meyer and Stewart W. Wilson, editors. From animals to animats : proceedings of the First International Conference on Simulation of Adaptive Behavior, volume 1, Paris, France, 1990. MIT Press, Cambridge, Mass.

[13] T.S. Ray. An approach to the synthesis of life. Artificial Life II, Santa Fe Institute Studies in the Sciences of Complexity, xi:371-408, 1991.