In each of the nine runs, the program length (and the replication period) of the programs still showed a net increase from the beginning to the end of the run, but the increments were fairly gradual. The smallest end-of-run dominant program length seen in any of the runs was 354 (Run 8), the largest was 402 (Run 1), and the average was 375. Plots of program length from four of the runs are shown in Figure 6.16. Analysis of individual programs revealed that the increase in size was due largely to the accumulation of energy collection instructions within the programs' copy loops.

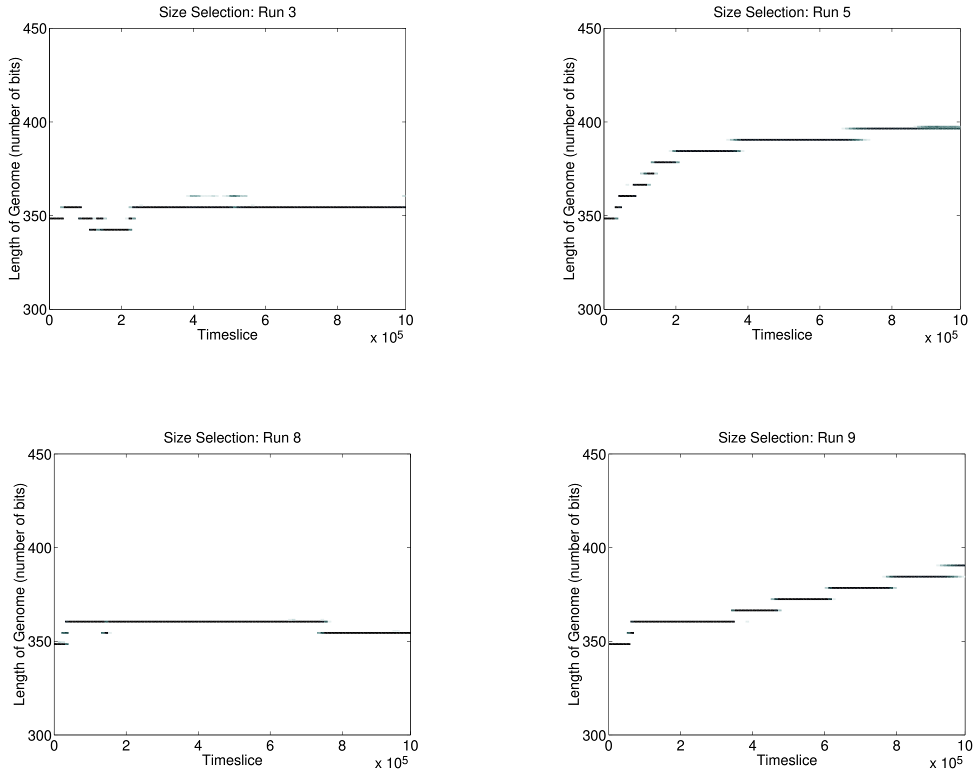

Figure 6.16: CPU-time Distribution: Graphs of Program Length for 4 of the 9 Runs.

6.5 Energy

From the experiments already discussed, it is clear that the distribution of energy in the environment, and its collection and storage by individual programs, are important factors in determining the behaviour of the system. In this section, a number of sets of experiments are reported in which various system parameters concerning energy distribution and collection are changed.

6.5.1 High Energy Levels

In the first set of energy experiments, the amount of energy introduced into the environment at each time slice was increased, as was the maximum amount of energy that any particular grid position could hold. The non-default parameter settings are shown in Table 6.6.

Parameter	*Value*
`number_of_energy_tokens_per_grid_pos_per_sweep`	100
`max_energy_tokens_per_grid_pos`	500

Table 6.6: Non-default Parameter Values for High Energy Experiments.

Results

The only major difference observed with these parameter settings was that the population size during Class 2 dynamics oscillated about the maximum of 2500, defined by the parameter `max_cells_per_process`, rather than in the lower region of 600–1000 individuals as seen in most other experiments. This was expected, as the higher energy levels allowed more programs to coexist. Runs 3, 4, 7 and 8 displayed Class 2 dynamics for their whole durations, whereas the other 5 runs switched to Class 1 dynamics at some point. The typical length of programs in these 5 runs during Class 1 dynamics was longer than observed in other experiments (ranging from approximately 1850 bits in Run 7 to 3800 bits in Run 2), which was again due to the increased energy levels.

6.5.2 Private Energy Collection

The inability of the population to continue evolving effectively in the Class 1 runs that we have encountered up to now seems to be due to the fact that a program's offspring cannot spread through the population. This is because the offspring are usually drained of energy by existing programs before they are able to reproduce, as discussed at the end of Section 6.2.1.

If we were to prevent programs from using energy stored in neighbouring programs, then we would expect that evolution might be able to continue even if the system were to enter a state resembling Class 1 dynamics. A collection of runs was conducted to test this, which used the non-default parameter value shown in Table 6.7.

Parameter	*Value*
`energy_collection_scheme`	`private`

Table 6.7: Non-default Parameter Value for Private Energy Experiments.

Results

Three of the runs (numbers 2, 3 and 7) displayed Class 2 dynamics for their whole duration. However, because programs in overcrowded areas were no longer being killed off by having their energy reserves drained, far more programs were able to coexist within the population at any given time. In fact, the population size oscillated around the limit of 2500 individuals imposed by the parameter `max_cells_per_process`, with a proportion (10%) of the population being culled each time this limit was reached (as determined by the parameter `population_cutback_on_overcrowding`).

Five of the other runs (numbers 1, 4, 5, 6 and 8) showed behaviour which looked similar to Class 1 dynamics. Graphs of some of the measures of interest are shown for Runs 4 and 5 in Figure 6.17.

Looking at these graphs, it is clear that the behaviour of these runs is not exactly the same as the Class 1 runs we have encountered before. Mean Cumulative Activity, $\bar{A}_{cum}$ (not shown), and Mean New Activity, $\bar{A}_{new}$, are both non-zero after the transition, the population size settles down in the region of 1500–2000 individuals, the diversity is lower than seen in previous Class 1 systems (but still higher than in Class 2 systems), the program length initially shows a marked increase, but in some runs subsequently became somewhat shorter again during the rest of the run, and the replication period shows a downward trend.

The fact that $\bar{A}_{cum}$ and $\bar{A}_{new}$ are non-zero suggests that particular individual genotypes *were* spreading throughout the population even after the transition, as expected (because their offspring were not being killed off by energy drainage before they had a chance to reproduce). This is also demonstrated by the appearance of significant waves in the activity wave diagrams (not shown) for these runs after the time of the transition. In other words, effective evolution is still possible, even after the transition, although it proceeds at a slower pace (the activity waves are smaller, and $\bar{A}_{cum}$ and $\bar{A}_{new}$ are lower after the transition). The slower pace is due to the increased competition for energy (because the longer programs are collecting more energy from the environment at each

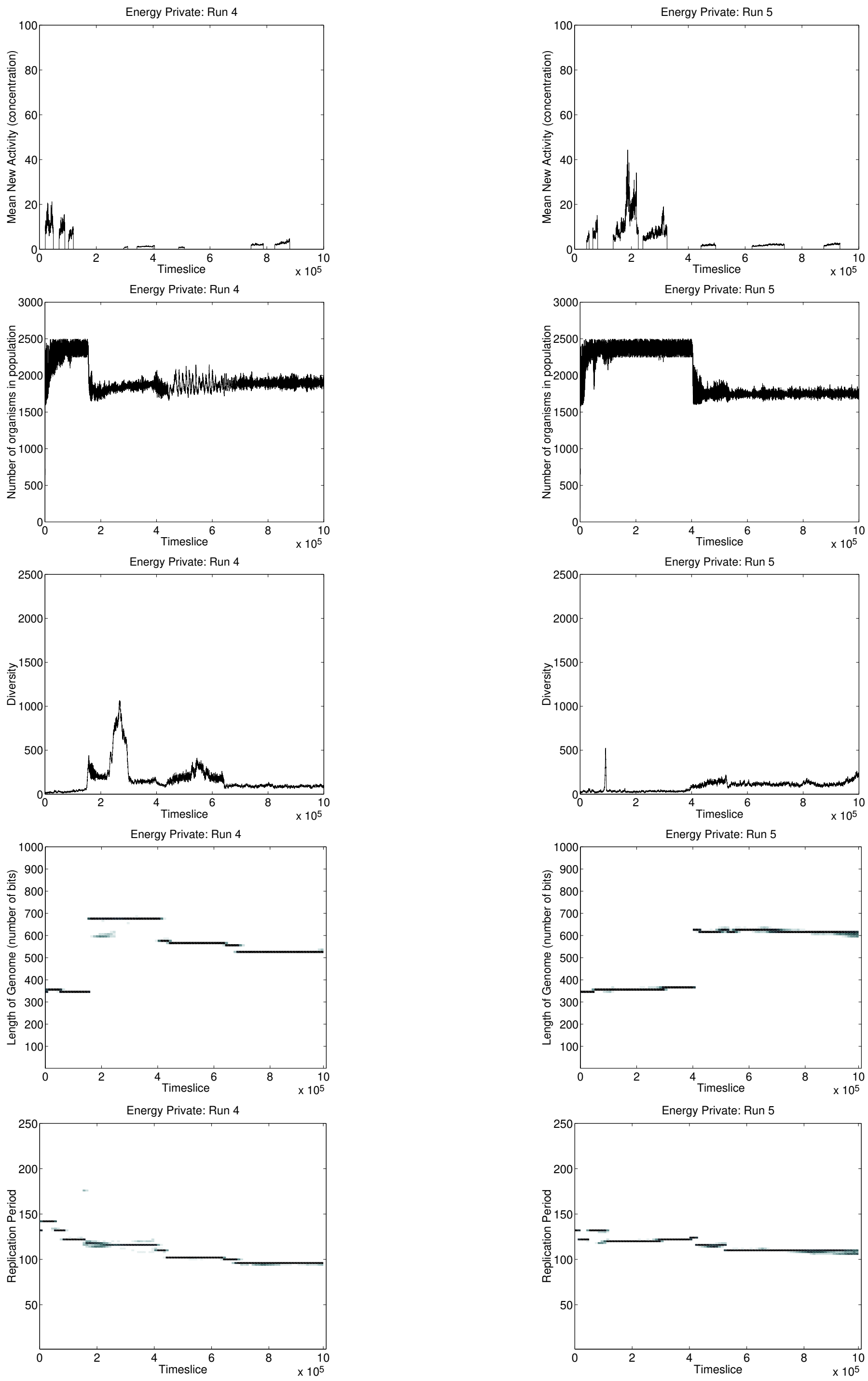

Figure 6.17: Private Energy Collection: Class 1-like behaviour. *Left*: Run 4, transition occurs at time slice $\sim 1.8{\times}10^5$. *Right*: Run 5, transition occurs at time slice $\sim 4.0{\times}10^5$.

time slice), which generally leads to shorter lifetimes for the programs. The somewhat smaller population size may also have contributed to the slower pace (but note that this itself is related to the greater competition for energy).

The fact that well-adapted genotypes can still spread throughout the population also explains why the diversity is lower than was observed in other Class 1 systems.

The decrease in program length in the period after the initial increase which caused the transition is explained by the fact that the system is still able to evolve. Analysis of individual genotypes in Run 4 showed that the shorter programs which emerged towards the end of the run had lost redundant instructions from within their copy loops, and were therefore able to reproduce faster (i.e. the replication period decreased). This process can be seen as a partial recovery of the system from the initial transition to Class 1 dynamics, bringing it back towards Class 2 dynamics.

The final run, number 9, showed results unlike any of the others. The most obvious differences were evident in the graphs of population size and diversity, shown in Figure 6.18.

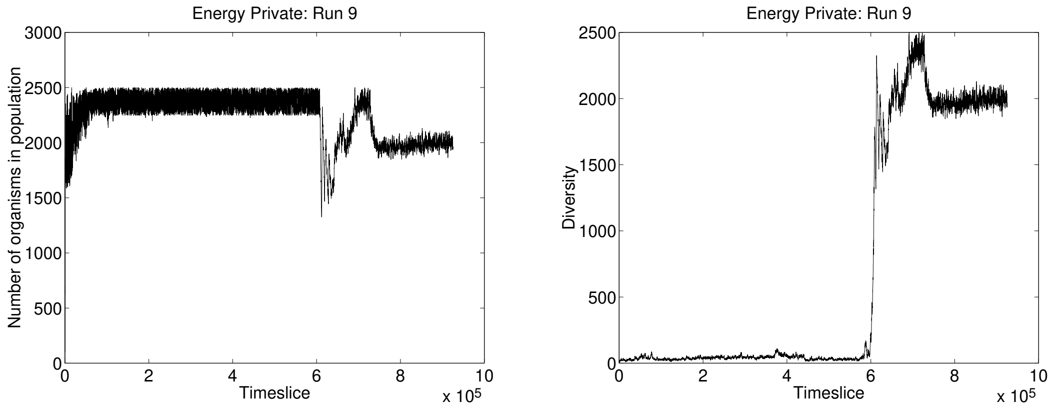

Figure 6.18: Private Energy Collection: Population Size *left* and Diversity *right* for Run 9.

Investigation of individual genotypes in this run revealed that the population had succumbed to a type a 'cancerous' mutation, as encountered in a previous experiment (see Section 6.3.2). As before, the effect of this mutation was that a program's offspring had an extra, redundant instruction tagged onto its end, so that the length of programs in the population grew by one instruction each generation. When we encountered this before, the effect was that the entire population eventually died out, because the programs were being drained of energy by executing all of the redundant instructions.

However, in the present case, it happened that the extra instruction that was being added to the end of the programs at each generation was an energy collection instruction (`et_collect`). As a consequence, rather than dying of energy shortage, the programs were collecting more and more energy as the 'cancer' grew. The programs therefore continued to live, although, as each one was so energy-hungry, the environment could only sustain a population size at the level of about one program per grid position (i.e. approximately 1600 individuals). Also, as each offspring was different from its parent (because it had an extra instruction at the end), the diversity of the population was very high, as shown in Figure 6.18. This mutation could therefore be described as a 'non-fatal cancer'.

6.5.3 Energy Gradient

In all of the experiments reported previously, energy has been supplied to the environment uniformly across the whole grid. In this section and the next, experiments are reported in which two alternative energy distribution schemes were used. In this section, the amount of energy given to each grid position varied in one direction along a linear gradient. This was achieved by using a non-zero value for the parameter `x_delta`, as described in Section 4.5.2. Two different gradients were investigated, as described in the following two subsections.

Small Gradient

The value used in these experiments is shown in Table 6.8. The effect of this value was that grid positions on the extreme left of the grid received 26 energy tokens at each pass through the main Cosmos control loop, those on the extreme right received 34, and those in between received intermediate numbers according to their position. Positions on the extreme right of the grid therefore received about 30% more energy than those on the extreme left. The mean number of energy tokens per grid position was 30, as in the standard run. It was hoped that a non-uniform environment such as this might lead to speciation, with different species specialising in living in areas with different availabilities of energy.

Parameter	*Value*
`x_delta`	0.2

Table 6.8: Non-default Parameter Value for Small Energy Gradient Experiments.

Results. No speciation was observed in these experiments. Three of the runs (numbers 3, 5 and 9) exhibited Class 2 dynamics for their whole duration, and five of the others (numbers 2, 4, 6, 7 and 8) switched to Class 1 dynamics during the run.

The remaining run (Run 1) probably fits into Class 1 as well, although the results were slightly unusual. Systems showing Class 1 dynamics typically contain properly functioning self-replicating programs, but as these are unable to spread though the population, numbers of each type of program are low, and the population diversity is high. However, in Run 1 of this experiment, *all* programs capable of faithful replication were lost after only 20,000 time slices. The population survived, but was composed of programs which either reproduced unfaithfully, or not at all. The evolutionary potential of this run had therefore been destroyed.

Analysis of the three runs which remained in Class 2 dynamics for their whole duration revealed that the ancestor program 348AAAA was able to survive even in the low-energy region, and that evolution proceeded in the same way as usual, with a single dominant genotype at any given time, towards greater numbers of energy collection instructions within the copy loop.

Large Gradient

In the light of the results obtained with a small energy gradient, the experiments were re-run using a larger gradient. The chosen `x_delta` value is shown in Table 6.9. Under these conditions, positions on the leftmost column of the grid received one energy token at each pass through the control loop, and those in the rightmost column received 59, with a linear gradient in between. The mean number of energy tokens per grid position across the whole grid was again 30.

Parameter	*Value*
`x_delta`	1.5

Table 6.9: Non-default Parameter Value for Large Energy Gradient Experiments.

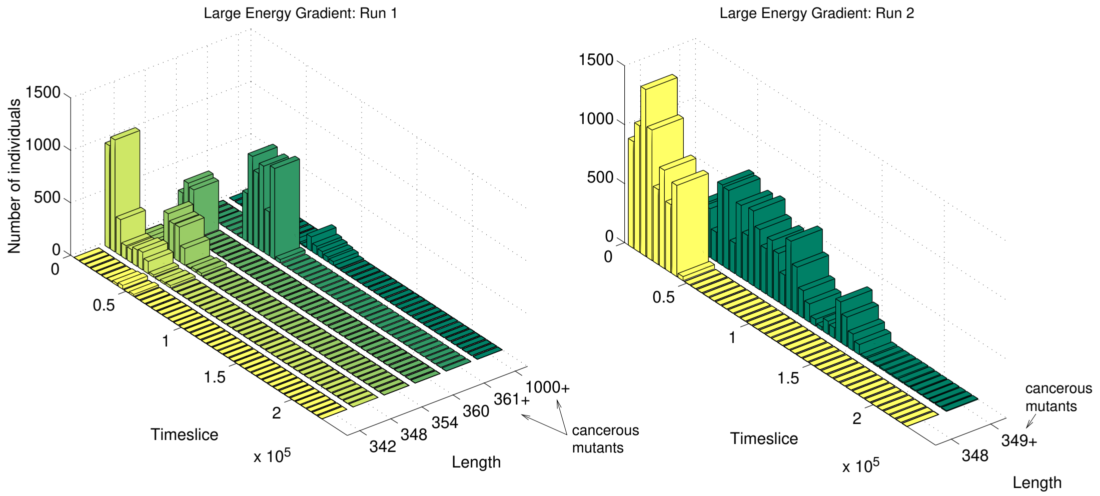

Figure 6.19: Large Energy Gradient, Runs 1 and 2: Change in constitution of population over time. In each graph, the number of individuals shown for each particular length is the sum of all programs in the population at that time which are instances of any genotype of that length. The bin labelled 1000+ in the left graph, and the bin labelled 349+ in the right graph, show data for all programs of the specified length *or longer*. The bin labelled 361+ in the left graph shows data for all programs with length in the range 361–999 bits.

Results. With the larger gradient, the population in two of the runs (numbers 1 and 2) was driven to extinction within the first 200,000 time slices. Of the other seven runs, graphs of the standard measures revealed that four runs displayed fairly standard Class 2 dynamics for their whole duration (Runs 3, 5, 7 and 8), while the other three experienced a transition to a state similar to Class 1 dynamics (Runs 4, 6 and 9).

A closer analysis of the two runs (numbers 1 and 2) where the population suffered extinction revealed that both had succumbed to a cancerous mutation early on in the run. The rise and fall of genotypes of different lengths in these runs in shown in Figure 6.19. In both runs, virtually all non-cancerous programs had disappeared within the first 50,000 time slices, and the whole population had become extinct by time slice 100,000 in Run 1, and time slice 200,000 in Run 2.

Investigation of the Class 2 runs revealed some interesting results. In two of them (Runs 3 and 8), the population was dominated by programs of a single length at the end of the run. However, in the other two (Runs 5 and 7), there were significant numbers of programs of more than one size: the end-of-run population in Run 5 contained programs of lengths 372, 378 and 384 bits; while the end-of-run population in Run 7 contained programs of lengths 366 and 372 bits. The change in the constitution of the populations of all four of these runs is shown graphically in Figure 6.20.

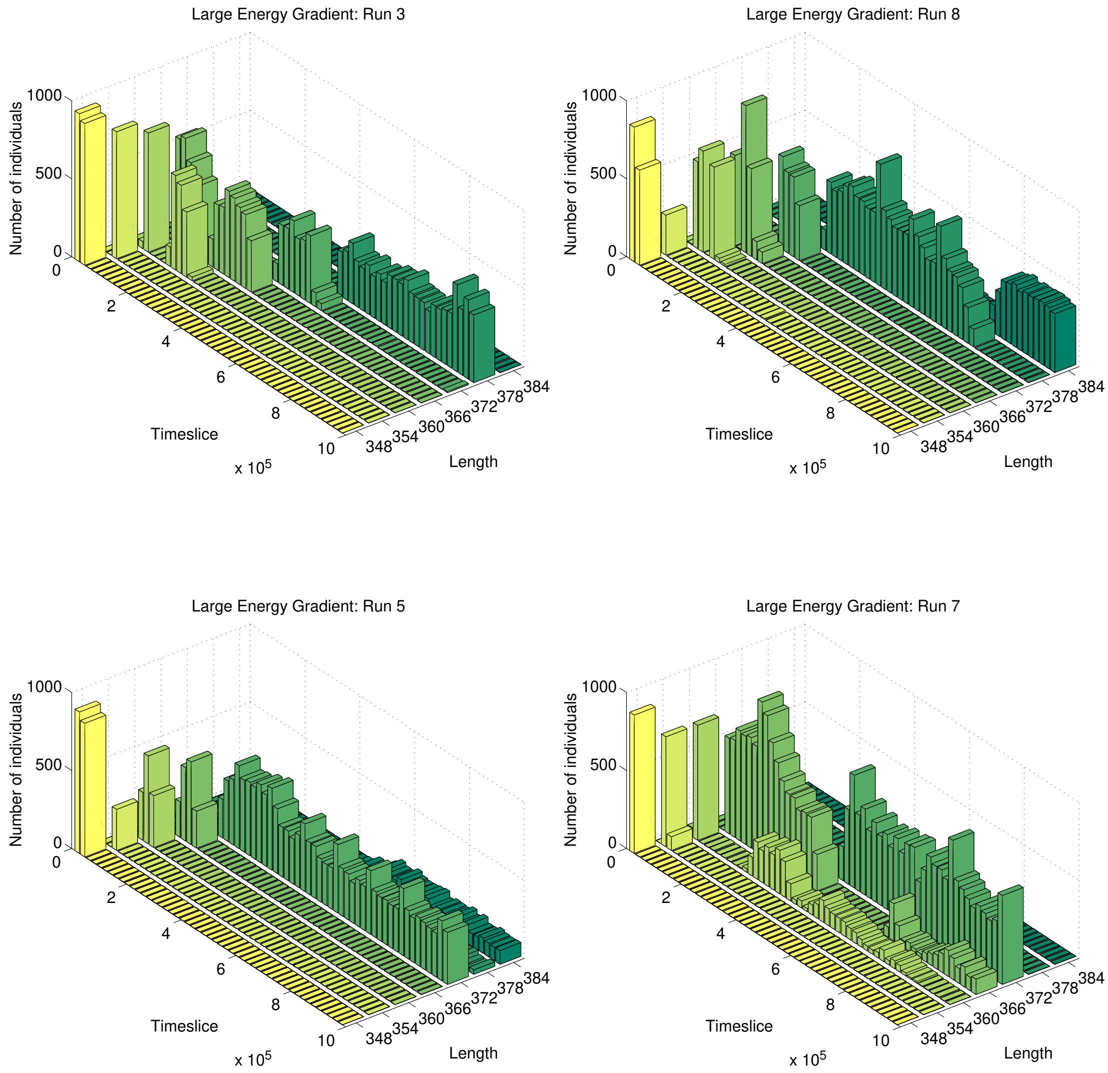

Figure 6.20: Large Energy Gradient, Runs 3, 5, 7 and 8: Change in constitution of population over time. In each graph, the number of individuals shown for each particular length is the sum of all programs in the population at that time which are instances of any genotype of that length.

These graphs further reveal that the population in Runs 3 and 8 tended to be composed of programs of a single dominant length at any point during the run. In contrast, the population in Runs 5 and 7 comprised programs of two or more different lengths for most of the duration of the run. This is the first experiment in which programs of different lengths have been observed to coexist for prolonged periods, at least while the system is in a Class 2 state.

In order to see whether these programs of different lengths were inhabiting different areas of the environment, graphs of spatial distributions were plotted for these four Class 2 runs.

For Runs 3 and 8, which contained programs of a single length only, the distributions are shown in Figures 6.21 and 6.22. This figures show that programs are occupying positions in the rightmost three-quarters of the grid. The reason why the leftmost quarter appears to be inhospitable can be understood by looking at the 'equilibrium' environmental energy distribution, plotted in Figure 6.23. This graph is plotted from data extracted from an output file, and shows how much energy is normally available in grid positions along the x-axis (i.e. along the energy gradient). For each position, this is the amount of energy which it will have in store if it has not been drained by programs in the recent past. The graph shows that positions in the leftmost quarter of the grid typically have fewer than 10 energy tokens available at any given time. Now, the ancestor program was designed on the assumption that each `et_collect` instruction would collect 10 energy tokens; if it regularly collects fewer than 10, it will eventually starve. The leftmost quarter of the grid in these experiments therefore contains insufficient energy for the ancestor programs to survive. The fact that no programs occupied these positions even towards the end of Runs 3 and 8 (and, as we will see, this is also true for all of the other runs) suggests that no programs evolved the ability to collect more energy from the environment and thus be able to survive in this region.

The shape of the graph in Figure 6.23 needs a little more explanation. The energy gradient on the left half of the grid is exactly as expected, and reflects the amount of energy distributed to those positions at each pass through the Cosmos control loop. The levels on the right half of the grid are somewhat unexpected. In these experiments, the parameter `max_energy_tokens_per_grid_pos` was set to its default value of

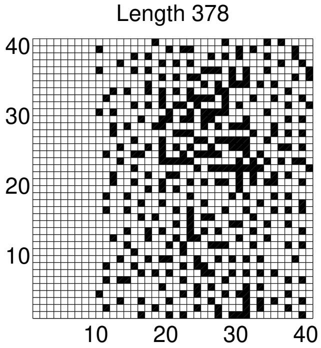

Figure 6.21: Large Energy Gradient, Run 3: Spatial Distribution of Programs at Time Slice 900,000. At this time, virtually all programs in the population were of length 378 bits.

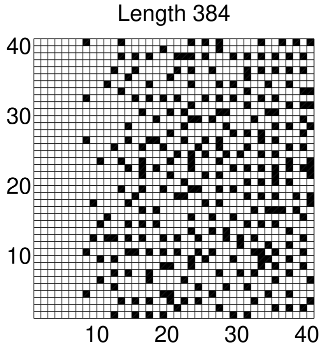

Figure 6.22: Large Energy Gradient, Run 8: Spatial Distribution of Programs at Time Slice 900,000. At this time, virtually all programs in the population were of length 384 bits.

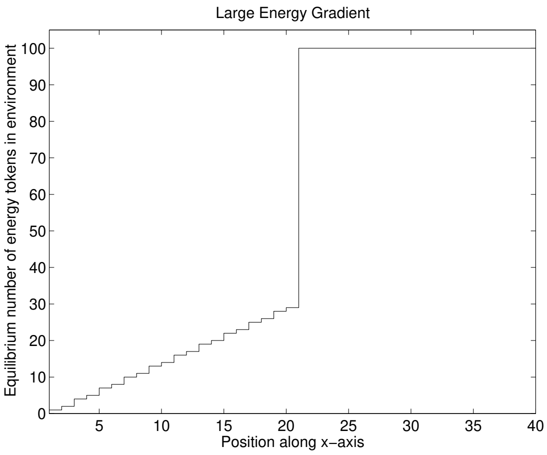

Figure 6.23: Large Energy Gradient: Equilibrium distribution of environmental energy.

100, and all positions on the right half of the grid store this maximum amount of energy (in the absence of programs to drain it). This turns out to be due to a feature of the `AttenuateEnvironmentalEnergy` routine, which is executed at the end of each pass through the main control loop (see Section 4.7). After all programs have had a chance to execute some instructions, this routine removes a certain amount of energy from each grid position in order to prevent it from building up excessively. The current implementation of this routine works by subtracting an amount determined by the parameter `number_of_energy_tokens_per_grid_pos_per_sweep` from each position. (The default value of this parameter, as used in the current experiments, is 30.) When energy is distributed evenly across the environment, this means that each position loses the same amount at the end of each iteration of the control loop as it is given at the beginning of the loop. However, in the current case, positions in the right half of the grid are given more than 30 energy tokens per iteration. They are therefore able to accumulate these tokens over time, and soon reach the maximum level allowed. This feature of the energy distribution turns out to have important consequences for the results of the other runs, as will now be explained.

The spatial distributions of programs in the two Class 2 runs which showed some differentiation in program lengths (Runs 5 and 7) are shown in Figures 6.24 and 6.25. These figures clearly show that programs of different lengths are generally located in different regions of the environment. The segregation is not straightforwardly related to program length itself, as in Run 5 the smaller programs are further to the right of the grid, whereas in Run 7 the opposite is true. Similarly, a comparison of the individual genotypes reveals that the segregation is also not straightforwardly related to the number of `et_collect` instructions that a program contains. In fact, comparison of the genotypes reveals that the only consistent difference between programs living in different areas is that those dwelling exclusively in the right half of the grid (i.e. programs of length 372 in both runs) retain the ability to *move*, where those living further towards the left of the grid are sessile.

Considering the fact that a quarter of the grid does not contain sufficient energy for programs to survive for long, being able to move (and potentially wander into this area) might not seem like a particularly advantageous capacity for the programs to retain. However, analysis of the programs' movements reveals that they have a very strong preference for moving upwards (Direction 4 in Figure 4.3(a)). In comparison, the number of programs that move in any other direction is negligible. Remembering that the environment wraps around in both directions in these experiments (i.e. it is really toroidal in shape), these programs can therefore move around in this manner, while always remaining in the right half of the grid. Figure 6.20 shows that the number of programs of length 372 is fairly high towards the end of Runs 5 and 7, so the right half of the grid is fairly crowded. (This is not so clear in Figures 6.24 and 6.25, because these do not distinguish grid positions occupied by more than one program from those occupied by a single program.) In this situation, it is conceivable that a program might do better on average by moving around, rather than risk being stuck at a location where it is competing for energy with one or more other programs.

In contrast, programs that live in the left half of the grid in these runs were sessile.

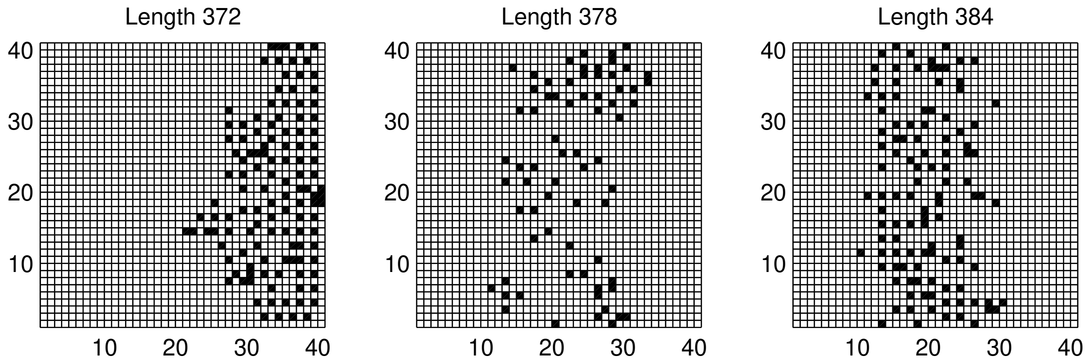

Figure 6.24: Large Energy Gradient, Run 5: Spatial Distribution of Programs of Different Sizes at Time Slice 900,000.

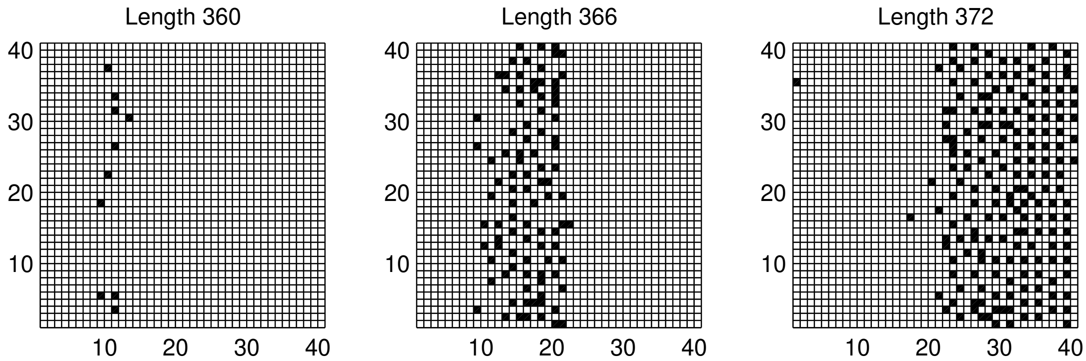

Figure 6.25: Large Energy Gradient, Run 7: Spatial Distribution of Programs of Different Sizes at Time Slice 900,000.

Grid positions in this half typically have much less available energy than those in the other half (Figure 6.23). The number of programs in this area is also much lower than in the other half (compare the number of programs of different lengths towards the end of Runs 5 and 7 in Figure 6.20 with the position of these programs in Figures 6.24 and 6.25). Under such conditions, these programs have adopted the strategy of staking a claim on a particular grid position. The concentration of programs in this area is low enough that they generally do not have directly neighbouring programs with which to compete for energy.

As a final point about these programs, notice that in Run 5, some of the sessile programs of lengths 378 and 384 are located in the right half of the grid, in competition with the motile programs of length 372. This suggests that the adaptive significance of movement in comparison to non-movement is not particularly great even in this half of the grid. The segregation of the different populations of programs has presumably been sustained by the existence of the energy gradient, but the initial appearance of the differences may have been due to rather contingent factors ('historical accidents'). This is supported by analysis of the programs in Runs 3 and 8, where no differentiation was seen. In Run 3, all of the programs were motile, and in Run 8 they were all sessile, despite the fact that in both runs the populations covered the full three-quarters of the environment (Figures 6.21 and 6.22).

Finally, the three runs which display Class 1-like dynamics (numbers 4, 6 and 9) were analysed. All three of these showed length differentiation by the end of the run, as shown in Figure 6.26. As is typical for Class 1 runs, the lengths of programs were much longer than in Class 2 runs. The difference in length between the longest and shortest species in these runs was also much larger than in the differentiated Class 2 runs we have seen already (Runs 5 and 7). Note that the overall population sizes towards the end of these runs are also considerably smaller than in the Class 2 runs, especially in Runs 4 and 9 (Figure 6.26).

The spatial distribution of programs of the dominant lengths at time slice 900,000 is shown for each of the three runs in Figures 6.27–6.29. Again, programs of different length are clearly generally located in different areas of the environment, for all three runs. However, analysis of the individual genotypes revealed that this time the segregation was not related to the ability or inability of programs to move; in Run 6, *all*

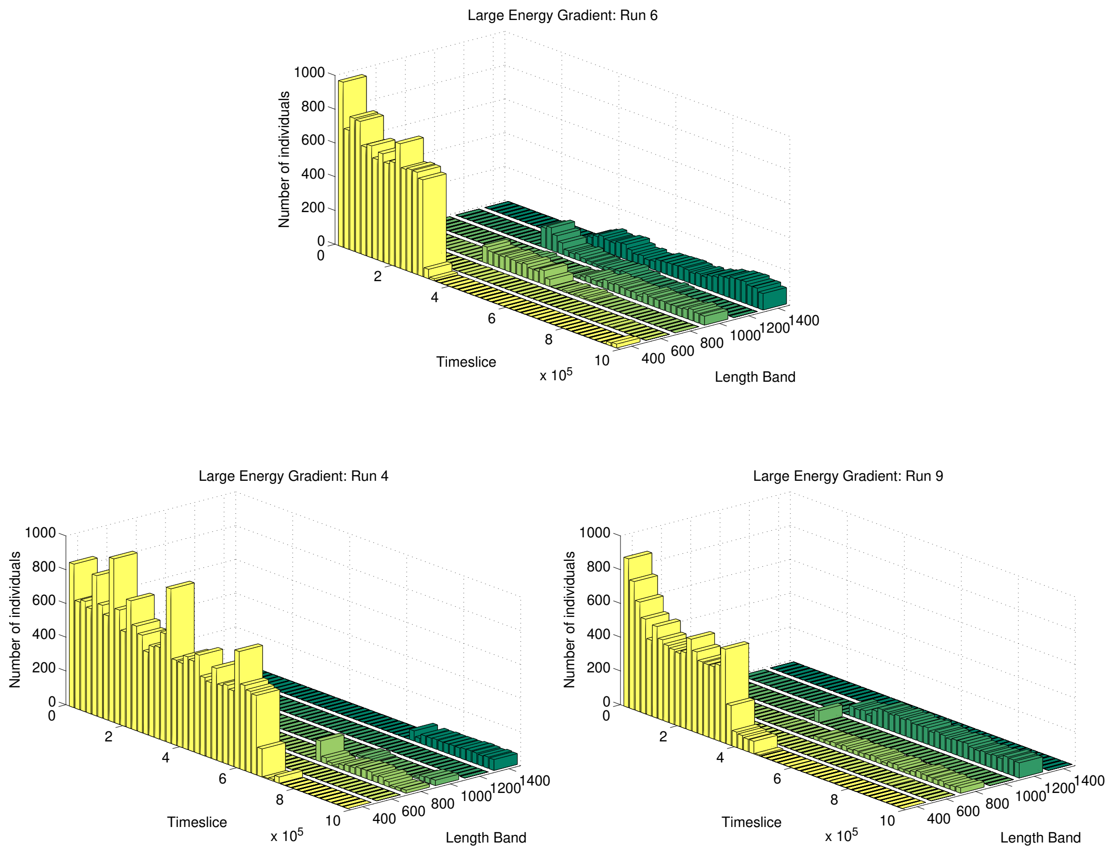

Figure 6.26: Large Energy Gradient, Runs 4, 6 and 9: Change in constitution of population over time. Each bin of the Length Band axis shows data for all programs in the population with length in a range of 200 bits, centred on the value marked. For example, the first bin, marked 400, shows the collected data of all programs of length 300–499 bits.

programs were sessile, and in Runs 4 and 9, *all* programs were motile, with a strong preference for moving upwards (Direction 4), or occasionally downwards (Direction 0). It appears that in these runs, the segregation was mainly due to the lengths of the programs. Grid positions in the area immediately to the left of the centre of the environment receive approximately 30 energy tokens at each pass through the control loop (Figure 6.23). This is the same amount that all positions normally receive under the standard parameter settings, and under those conditions (i.e. in most of the other experiments reported in this chapter) we have seen that the lengths of Class 1 programs are generally in the range 600–1200 bits. In the present runs, we see that programs in this central region also have lengths in that range (Figures 6.27–6.29). In contrast, programs to the right of the environment are of greater length (typically 1250–1500 bits). Remembering that longer programs are allowed to execute more instructions per time slice, and therefore are potentially able to collect more energy per time slice, the location of these programs can be explained by the larger amounts of energy typically available in the right half of the environment.

6.5.4 Random Distribution

Another way in which the distribution of environmental energy can be made non-uniform in Cosmos is to have it distributed randomly. Under this scheme, chunks of energy tokens are distributed to grid positions at random, until the total amount has been allocated.[4] If the mean number of energy tokens distributed to each grid position was the same as in the standard run (i.e. 30 per time slice), then the populations soon died out. The mean number was therefore increased by threefold to 90 energy tokens per time slice, and the maximum number of energy tokens that any grid position could store was also increased somewhat. Under these conditions, the populations could often survive for reasonable durations. The non-default parameter settings for these experiments are shown in Table 6.10.

[4] Recall from Chapter 4 that under this scheme, the parameter `number_of_energy_tokens_per_grid_pos_per_sweep` determines the *mean* number of energy tokens that should be distributed to each grid position.

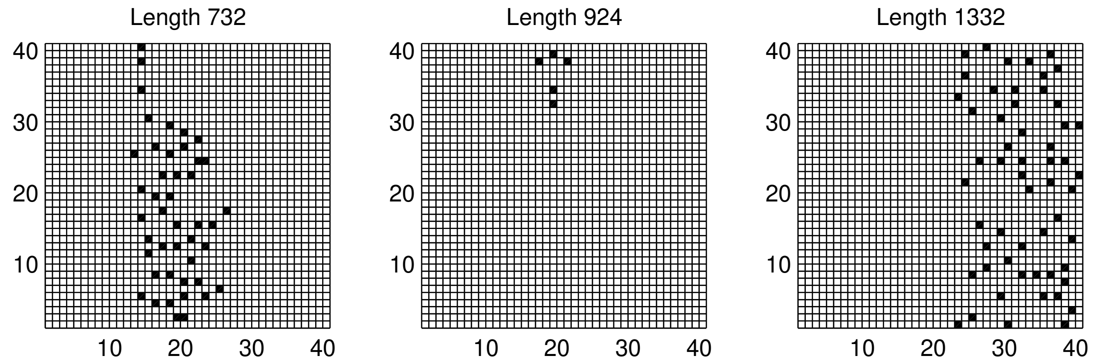

Figure 6.27: Large Energy Gradient, Run 4: Spatial Distribution of Programs of Different Sizes at Time Slice 900,000.

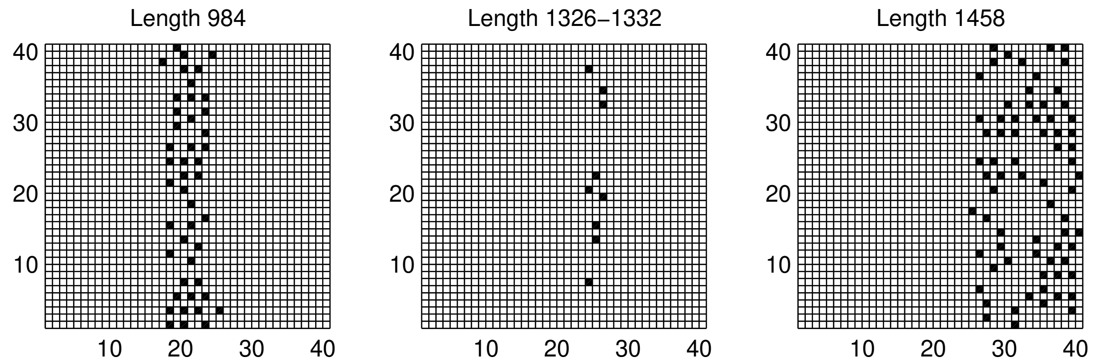

Figure 6.28: Large Energy Gradient, Run 6: Spatial Distribution of Programs of Different Sizes at Time Slice 900,000.

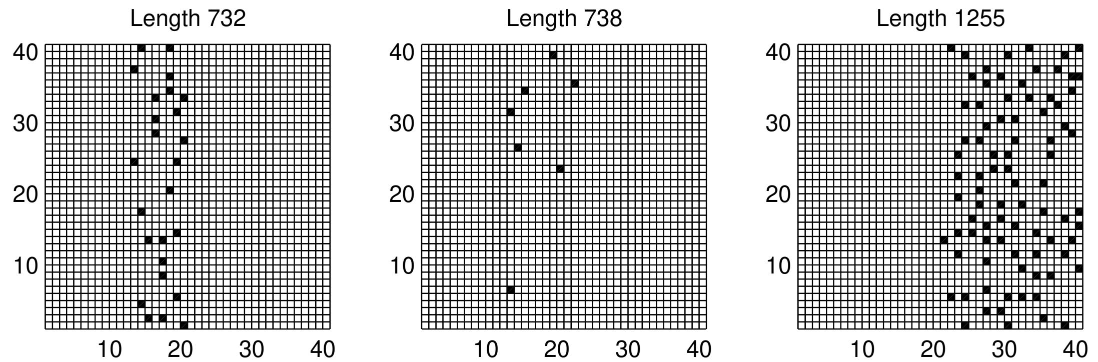

Figure 6.29: Large Energy Gradient, Run 9: Spatial Distribution of Programs of Different Sizes at Time Slice 900,000.

Parameter	*Value*
`number_of_energy_tokens_per_grid_pos_per_sweep`	90
`max_energy_tokens_per_grid_pos`	150
`energy_distribution_scheme`	random
`energy_distribution_random_chunk_size`	30

Table 6.10: Non-default Parameter Values for Random Energy Experiments.

Results

Under these conditions, the population survived for the duration of the run in six out of the nine experiments. Of the three in which the entire population died out (Runs 2, 3 and 9), the extinction occurred fairly early in the run in two cases (by time slice 78,600 for Run 2, and time slice 141,300 for Run 3), but at the very end of the run in the other case (Run 9, time slice 971,247).

Of the six which persisted for the whole run, three maintained Class 2 dynamics for the duration (Runs 1, 4 and 6), and three switched to Class 1 dynamics before completion (Runs 5, 7 and 8). The results were generally very similar to the standard runs, but there are a few points to note.

One difference was that programs retained the ancestral ability to move around the environment—an ability which is generally lost in the other sets of experiments reported. Indeed, analysis of the dominant end-of-run programs in the Class 2 runs (1, 4 and 6) revealed that they had all actually gained at least one more `move` instruction[5] somewhere in their code (but, interestingly, not within the copy loop). The explanation for retaining this capacity for movement is presumably because there is at least a chance that a nearby position will be rich in energy, while a position at which the program has been situated for some time will generally have fairly low energy reserves.

Another difference was that the population sizes in the runs were generally somewhat larger than usual, at least while they were in the Class 2 state. This is attributed to the greater mean energy levels in the environment.

Finally, in two of the three runs which switched to Class 1 dynamics (Runs 7 and 8), the population size in the Class 1 state was lower than normal (fewer than 150 organisms), and the diversity was also lower (approximately 20 different types of program). Analysis

[5] Or sometimes a `migrate` instruction, which is equivalent to `move` for a single-celled program.

of the programs in the population at the end of these runs shows that they had retained the ability to move, as in the Class 2 runs. This is in contrast to Run 5, which displayed more typical Class 1 dynamics, and in which the programs had lost the ability to move. Remember from Section 6.2.1 that Class 1 dynamics come about when programs grow so long that they drain energy from their neighbouring programs to the extent that the neighbours die. The stable configuration shown in Figure 6.10 allows for the maximum number of programs under such conditions. As the programs are motile in Runs 7 and 8, this stable configuration is not possible, so the population size is inevitably lower. However, another consequence of the programs' movement is that their offspring are distributed all around the environment rather than all being born in the same locality. This may provide a program with a slightly better chance than it would have were it sessile of propagating throughout the population (because at least *some* of its offspring might be born in favourable positions). In other words, the evolutionary potential of these runs has not been completely lost, which explains the lower diversity than is normal for Class 1 systems.

6.6 Reading Neighbouring Code

Programs in Cosmos cannot generally read the code of neighbouring programs. In contrast, programs in Ray's Tierra system do have this ability, and this is why parasites and other sorts of ecological interactions emerge fairly easily in it. A mechanism was built into Cosmos to simulate the reading of neighbouring code using the program's general communications input and output apparatus, as described in Chapter 4 (Sections 4.3.7 and 4.6.1). However, the mechanism is still more complicated than in Tierra, and would require somewhat more evolutionary innovation in order to be utilised effectively. A set of experiments was conducted in order to test the mechanism and see whether parasites could evolve in Cosmos. The non-default parameter setting is shown in Table 6.11.

Parameter	*Value*
`neighbouring_genomes_readable`	`yes`

Table 6.11: Non-default Parameter Value for Readable Neighbouring Code Experiments.

Results

No parasites or similar phenomena were observed in any of the nine runs. It appears that the mechanism used to simulate the reading of neighbouring programs' code was too complicated to be exploited by evolution. In particular, unlike in Tierra (see Section 3.3.1), there is no single mutation of the ancestor which would produce a functional parasite.

An analysis of the activity of the most numerous genotypes in each of the runs shows that they generally produced a much larger proportion of unfaithful offspring than did genotypes in the standard run. For example, the data recorded in the output file `morgue.dat` shows that the mean number of unfaithful reproductions for programs of the ancestor genotype 348AAAA was 0.277 (with standard deviation 0.099) across all nine runs. This compares to a mean of 0.010 unfaithful reproductions (with standard deviation 0.003) in the nine re-runs of the standard model, in which neighbouring code was not readable.[6]

This decrease in the copy fidelity of the programs was evidently caused by the mechanism for reading neighbouring code. Recall that this mechanism will attempt to bind a promoter onto the code of every available neighbour before it tries the host code. In eight of the nine runs, a state resembling Class 1 dynamics quickly arose, with the appearance of programs with length of order 1000 bits, and population sizes of order 400 individuals. The high proportional of unfaithful offspring produced by programs meant that the evolutionary potential of the runs was limited right from the start, and no trends for adaptive improvements of any sort were observed. In the other run (Run 7), the population was reduced to a single program by time slice 87,600, which survived until the end of the run.

In summary, the mechanism for reading the code of neighbouring programs failed to reproduce the results observed in Tierra. In particular, no parasites emerged during these runs.

[6] In contrast, the mean number of *faithful* reproductions of programs of the ancestor genotype 348AAAA were similar in the two sets of experiments. In the present experiment, the figure was 1.077 (standard deviation 0.045), and in the standard model re-runs, it was 0.993 (standard deviation 0.037).

6.7 Inoculation with Sexual Ancestors

All of the experiments reported so far have used the same hand-written self-replicating program, 348AAAA, as the inoculated ancestor. This program is fairly simple, and reproduces asexually, as described in Section 5.2.

In this final results section of the chapter, a set of experiments is described in which a more complicated ancestor program, 1314AAAA, was used. The most significant difference between this program and 348AAAA is that 1314AAAA reproduces *sexually*. The program is shown schematically in Figure 6.30, and a full listing is given in Section B.3.

The basic operation of the program is as follows. Most importantly, the program is divided into two sections (male and female). Under normal operation, any particular individual will only ever execute one of these two sections. The choice of which section it executes (i.e. the sex of the individual) is determined by a promoter; two different promoters are used, one specifying a male, and the other a female.

The female section works in a fairly similar fashion to the usual ancestor 348AAAA. The difference is that rather than copying the genome to the Nucleus Working Memory, it copies it to the Communications Working Memory. When the whole genome has been copied, the female then emits the copy into the environment as a message. At this point, the female's work is done, and it attempts to repeat this procedure indefinitely.

The male section begins with a section of code which causes the organism to wander around the environment for a predetermined duration, collecting energy.[7] The male then enters a behavioural loop, in which it seeks messages in the environment. During this phase, the male is sessile, but it looks for messages in all directions. It also checks its internal energy levels, and, if these are low, attempts to collect more energy from the environment. This phase continues until a message has successfully been recovered from the environment. When this occurs, the message is transferred to the male's Received Message Store. The start and end addresses of the message are calculated, and it is copied, one instruction at a time, into the male's Nucleus Working Memory. When the copying is complete, a calculation is performed to determine the sex of the

[7] This section was included for the practical reason that when a run is first inoculated, several hundred time slices pass before the females emit their first messages. During this time, collecting energy is the only useful task that the males can do.

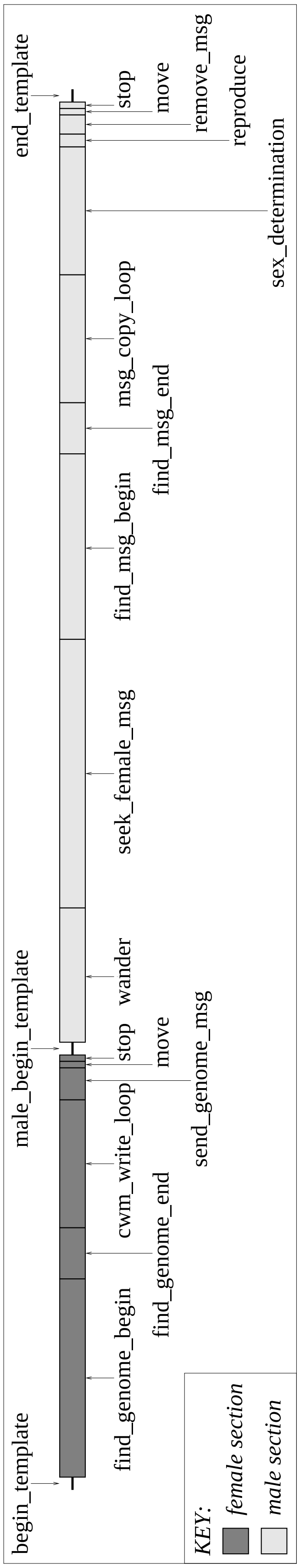

Figure 6.30: Schematic of the Sexual Ancestor, 1314AAAA.

offspring which is about to be produced. This is determined according to whether the male's internal energy store contains an odd or even number of energy tokens (i.e. sex determination is pseudo-random). Depending on this choice, an appropriate promoter is then manufactured for the offspring. At this stage, the offspring is finally produced. As always, the contents of the parent's Nucleus Working Memory becomes the genome of the offspring. The male parent finally removes the message from its Received Message Store, then repeats the whole process again.

To summarise, the whole genome is initially copied by the female, who transmits the copy as a message into the environment. The males attempt to pick up these messages. When they have done so, they produce an offspring whose genome is a copy of the received message. In this way, both males and females are required to propagate the species.

In the following experiments, half of the inoculated programs were given a male promoter, and the other half a female promoter. Males and females were placed in alternate positions in both directions, so they were regularly interspersed. The other non-default parameter settings used for these experiments are shown in Table 6.12. These values were arrived at after a series of trial-and-error attempts to reach a state where the population of programs in most runs survived for a reasonable length of time.

Parameter	*Value*
`number`	400
`rms_receive_search_area`	24
`energy_collection_scheme`	`private`
`et_value_constant`	10
`et_value_power`	0
`number_of_energy_tokens_per_collect`	15

Table 6.12: Non-default Parameter Values for Sexual Reproduction Experiments.

Results

Analysis of the output data revealed one basic result; the capacity for sexual reproduction was lost at a very early stage in all nine runs. By studying some of the individual genotypes from the runs, it was apparent that a minor mutation in the male section of the code responsible for collecting female 'spores' from the environment (the section labelled seek_female_msg in Figure 6.30) was sufficient to turn the male into a fully functional, independent, asexual organism. Mutations of this sort led to the sexual organisms being completely displaced by asexual organisms within the first 90,000 time slices in all runs, and indeed much earlier in most of them.

After the transition to asexuality had been made, subsequent improvements in the organisms were possible by the loss of the now redundant female section of the genome, and also by the loss of redundant sections of the male section (mostly within the areas labelled wander and seek_female_msg in Figure 6.30). The state of the system at the end of each of the runs can be classified as follows: in Runs 5 and 8, no progress had been made in removing redundant code from the asexual programs—they were still of the same length as the original sexual ancestor; in Runs 1, 2, 6, 7 and 9, most of the female section of the code had been removed from the programs, leading to programs with lengths in the range of 936–1098 bits (although in Run 7 this improvement only occurred within the final 50,000 time slices); and in Runs 3 and 4, parts of the redundant areas of the male section had been removed in addition to most of the female section, leading to programs with lengths in the region of 580–670 bits.

A final point to note is that in the programs alive at the end of Run 4, there was also evidence that a cancer had invaded the population. This was of the type where the extra instructions accumulating at the ends of the programs were energy collection instructions, as observed in a previous run discussed in Section 6.5.2. It is interesting that in both of the experiments in which this phenomena was observed, the parameter `energy_collection_scheme` was set to `private`. Recall that this means that programs can only extract energy from the (abiotic) environment, and not from their neighbouring programs. If such a cancer were to arise in a run in which programs could extract energy from their neighbours, the system would soon fall into Class 1 dynamics, as the amount of energy extracted would soon rise to a level where all such neighbours were killed. It is only when programs are prevented from extracting energy from their neighbours that this cancer can develop to the extent that it did in Run 4 of the current set of runs, and in Run 9 of the Private Energy Collection runs reported in Section 6.5.2.

6.8 Summary and Discussion

To end this rather long chapter, it might be useful to recap the more important results, and to discuss their significance. As well as reporting the experiments which have been performed, it is also important to consider what types of experiment might also be possible but are yet to be attempted. A number of possible lines of future experimentation are suggested at the end of the section.

Perhaps the most interesting result to come out of this research has been the analysis of the role of contingency in determining the course of evolution. In Section 6.1 we discovered that in a set of 19 runs of Cosmos which differed only in the number used to seed the random number generator, each one performed significantly differently, in a number of measures, from at least one third of the other runs. Using these results, it was decided that each Cosmos experiment in future should be run at least nine times in order to be confident of seeing a variety of results.

Despite the significant quantitative differences that arise, by chance, between many of the runs, we have seen that the system's behaviour can generally be categorised into one of two classes proposed by Bedau and colleagues [Bedau *et al.* 98]. This finding is reminiscent of the evolutionary dictum of "laws in the background and contingency in the details" discussed in Chapter 2 (Section 2.3.5), although in the present case even the background laws would appear to be rather dependent upon the details of the system's design.

During Class 2 dynamics, the population of programs is able to evolve fairly successfully. The chief methods by which adaptive improvements are made are by the accumulation of `et_collect` instructions within the programs (enabling them to collect more energy from the environment), and by the removal of redundant instructions. Note, however, that there is a limit to the extent that each of these methods can be applied; in Section 6.3.1 we saw that as the number of `et_collect` instructions in a program's copy loop was increased, the system moved towards Class 1 dynamics, where evolution effectively ceases; also, there is obviously a limit to the number of redundant instructions that can be removed from the programs. Once these limits have been reached, another kind of adaptive innovation would have to be discovered to allow the system to continue evolving, but we have so far seen little evidence for other innovations of this nature.

The other category of behaviour that was often observed was Class 1 dynamics. Analysis of runs which had switched to this state revealed that it is degenerative from an evolutionary point of view, because adaptive innovations are no longer able to spread through the population. The transition from Class 2 to Class 1 dynamics is usually brought about by the appearance of programs much longer than the ancestor (typically in the range of 600–1200 bits). Under the standard parameter settings whereby CPU-time is allocated to programs according to their length, longer programs such as these are able to collect more energy from the environment per time slice, giving them an advantage over their shorter ancestors. The onset of Class 1 dynamics can be avoided by using a length-neutral CPU-time allocation scheme (Section 6.4). The evolutionary activity of the system in the Class 1 state can also be improved by using a private energy collection scheme (Section 6.5.2).

Another interesting result was the speciation[8] of programs of different lengths, observed in the experiments in which a large gradient existed in the distribution of environmental energy across the grid (Section 6.5.3). This result suggests that a heterogeneous environment is an important factor in the emergence of speciation, and also demonstrates that it is not necessary to have a physical boundary for the population to differentiate into a number of species. In biological terms, the origin of species which live in the same place is called sympatric speciation. A number of processes have been proposed to explain this phenomenon, but the mechanisms involved are usually somewhat more complicated than the simple gradient in resource availability involved in these Cosmos runs [Maynard Smith 89]. Although the experiments reported here cannot really tell us much about this issue,[9] they do at least demonstrate the potential of spatially explicit, individual-based models to investigate such questions.

The experiments with sexually reproducing ancestors reported in Section 6.7 demonstrated that this capacity is easily lost. In all of the runs, the sexual ancestors were rapidly displaced by asexual programs. Further analysis revealed that this may have

---

[8] I am using the term 'speciation' very loosely here. For a start, as the programs are reproducing asexually, it is debatable whether the concept of species is applicable at all. Even if we do allow the term to be used, it might be argued that the programs in the different length groups are fundamentally very similar, and are more appropriately viewed as different varieties of the same species. However, remember that they did not only differ in their lengths, but also sometimes in their ability to move, and by the amount of energy they collected.

[9] They were not designed with this in mind. A proper test would need to be based upon an explicit set of assumptions, and would almost certainly require sexually reproducing programs.

been partially due to the ease at which the sexual ancestor could be mutated into a functional asexual program. However, the results still suggest that it might be hard to devise an environment in which sexual programs have a selective advantage.

Finally, another phenomenon which arose on a number of occasions was the so-called 'cancer' mutation (Sections 6.3.2, 6.5.2, 6.5.3 and 6.7). This mutation resulted in an extra instruction being tacked onto the end of a program's offspring at each reproduction. Over the generations the size of this tumour therefore grew larger and larger, resulting eventually in the death of the programs and the extinction of the population (unless the tumour happened to be composed of energy collection instructions). Although this result bears little relation to cancer in biological organisms, it does at least demonstrate how mutations which are harmful to the population in the long-run can initially be selected for and thereby invade the population.

Two things which we have *not* observed in evolutionary runs with Cosmos are the appearance of parasites or similar symbiotic phenomena, and the appearance of multicellular (parallel) programs.

The fact that parasites did not evolve in Cosmos, unlike in Tierra, was anticipated. Their emergence in Tierra depends upon some fairly specific aspects of the system's design and of the ancestor program used, as discussed in Chapter 3 (Sections 3.2.1 and 3.3.1). As the design of Cosmos differed from Tierra in these respects, it is much harder for evolution to 'discover' parasites in Cosmos. This highlights the importance that seemingly innocuous design features might have in determining the behaviour of artificial life platforms such as these, and suggests that we should take great care when claiming anything about the generality of results obtained from any particular one.

Recall from Chapter 4 that one of the original reasons for building Cosmos was to create a platform in which the emergence of multicellular programs might occur fairly easily. However, the experiments reported in this chapter have been concerned with exploring rather more basic aspects of the system's design, and we have not yet had the opportunity to look into the issue of multicellularity in a systematic way. As a passing note, developmental biologist Lewis Wolpert has suggested that multicellular organisms might originally have emerged in conditions where food was sparsely distributed in the environment.[10] When no food was available, a multicellular organism would be able to begin eating its own cells to survive until environmental food was available again. The experiments with environmental energy gradients and with random energy distribution reported in this chapter could be seen as a recreation of such a scenario, but no multicellularity was observed in them. However, a much more thorough investigation into this topic is required before anything can be said on the matter.

Although this chapter has been rather long, we have really only just scratched the surface of exploring the parameter space. We cannot claim to have thoroughly investigated the parameter space for those parameters discussed in this chapter, and indeed we have not even begun to investigate the effect of varying the parameters listed in Section B.1. Many of these concern the mechanisms for dealing with environmentally-conveyed messages, and various factors involved with multicellular (parallel) programs. The sexual ancestor reported in Section 6.7 did use environmentally-conveyed messages, but this ability was rapidly lost during the course of evolution.

All of the experiments except those reported in Section 6.7 used the same ancestor program, 348AAAA. There is an infinite variety of other ancestors which could have been used, some of which would have undoubtedly led to very different results. For example, one of the theories which has been put forward to explain the sudden origin and rapid explosion in diversity of multicellular biological organisms during the late Precambrian sees the evolution of heterotrophs (i.e. organisms which eat other organisms in order to obtain the complex organic compounds they require for metabolism) as the prime cause [Stanley 73]. Such a scenario could be recreated in Cosmos by introducing heterotrophic organisms (modelled as programs which kill other programs to steal their energy reserves),[11] in an effort to bring about a similar explosion in diversity. Indeed, initial experiments in this direction have been attempted, but it has so far proved impossible to engineer a viable community that includes heterotrophic programs.

A final feature of the model which has received little attention is the mapping between a program's genome (represented as a string of binary digits), and the instructions in

[10] This theory, which he named 'cannibalistic altruism', was discussed during a talk by Wolpert at the Royal Museum of Scotland, Edinburgh, on 20 February 1997.

[11] But note that in this case the prey organisms would only be a resource of energy, not of matter. This distinction is discussed in more detail in the next chapter (e.g. Section 7.1.4), along with possible evolutionary consequences arising from it.

the language provided. At present, there are 62 instructions in the Cosmos instruction set, and these are encoded using six bits (giving a total of 64 different possibilities). All of the runs reported in this thesis have used the same mapping, in which each of the 62 instructions is represented once, except the `move` instruction which appears three times (see Section B.4). There is therefore virtually no redundancy in the encoding, in contrast to the biological genetic code which encodes 20 amino acids with 64 possible codons. An alternative design has been developed, which uses a reduced instruction set. This consists of just 21 primary units which can be encoded on the genome. The full functionality of the existing system is maintained by allowing the primary units to form compound instructions. This is somewhat analogous to the way in which biological genomes encode just 20 amino acids, which, when decoded, are then assembled into a vast array of useful proteins. It may turn out that such redundancy would have a considerable effect upon the evolvability of the system. For example, important or frequently-used instructions in the language could be represented by multiple codons, thereby biassing the system to use these rather than some of the more obscure instructions. However, this design has so far not been implemented.