Chapter 6

Cosmos Experiments 2: Exploring the Parameter Space

The previous chapter described the results of a single Cosmos run. In the present chapter we will first investigate the extent to which the differences in behaviour between a number of runs are due purely to the stochastic nature of the system (Section 6.1). This gives us an idea of how many times we should run each experiment, keeping all parameter values constant, to see a range of different behaviours arising solely through this stochasticity. With this in mind, the results of a collection of further experiments are then described, which were designed to explore the parameter space of the system somewhat. For all of the experiments reported, any non-default parameter settings are listed; most are specified in the relevant sections of this chapter, but the values of the parameter `rng_seed`, used to seed the random number generator for each of the runs, are listed separately in Section B.5.

6.1 The Role of Chance[1]

Some of the results reported in the previous chapter suggest that chance events may have played a significant role in the outcome of the run (see Section 5.2.7). As discussed in Section 2.3.5, it is generally accepted that contingency ("historical accident") has played a large part in determining the course of evolution on Earth, although there is still much debate as to the relative importance of chance events versus other factors

[1] This section is based upon a previously published paper, [Taylor & Hallam 98].

(e.g. the inherent self-organisational properties of the biochemical world, discussed in Section 2.1.1). With artificial evolutionary systems we have the advantage of being able to "replay" evolution under experimental control. When considering the performance of an evolutionary system, we generally wish to disentangle the relative influence of three factors: (1) contingency, (2) performance due to particular details of the system's design, and (3) performance which may be general to a wide class of evolutionary systems [Taylor & Hallam 97]. In this section, an experiment is reported in which Cosmos was run a number of times, varying just the random number seed between runs. The results will give us a better idea of the role of contingency in the system.

6.1.1 Method

Nineteen runs of Cosmos were initialised, each with exactly the same ancestor programs, and exactly the same parameter values except for the seed for the random number generator (RNG). Most parameters took on their default values; those that did not are listed in Table 6.1.[2] For each completed run, the following measures were investigated:

1. Program age at death
2. Replication period (time between 1st and 2nd faithful replication)
3. Program length
4. Flaw period
5. Number of faithful replications per program
6. Number of unfaithful replications per program
7. Population size
8. Population diversity
9. Cumulative activity
10. Mean cumulative activity
11. Activity wave plots

[2] These experiments were run on slower machines than most of the others, which is why the size of the runs (i.e. the maximum number of cells allowed in the population and the run duration) is smaller than most of the others. The other differences in parameter values came about because these runs were actually conducted before most of the others, and the default values of a few of the parameters were changed in the intervening time. The differences are only minor, and it is not expected that they affect the applicability of the results to the other experiments reported in this chapter.

Parameter	*Value*
inoculation	
`number`	`64`
startinfo	
`rng_seed`	[*variable*]
termination	
`number_of_timeslices`	`300000`
environment	
`max_cells_per_process`	`800`
`number_of_energy_tokens_per_grid_pos_per_sweep`	`10`
`max_energy_tokens_per_grid_pos`	`25`
`energy_collection_scheme`	`private`
`x_delta`	`0.025`
cell	
`max_energy_tokens_per_cell`	`50`
`neighbouring_genomes_readable`	`yes`
mutation	
`mutation_application_period`	`1`
`default_flaw_period`	`1000000`

Table 6.1: Non-default Parameter Values for Contingency Experiments.

6.1.2 Results

For each measure, the results from each of the 19 runs were compared.[3] Some of the measures (Population Size, Age at Death, Flaw Period, Number of Faithful Replications and Number of Unfaithful Replications) generally showed no trends, and their absolute values were very similar across different runs. In contrast, trends *were* observed in the five other measures (i.e. Cumulative Activity, Mean Cumulative Activity, Diversity, Program Length and Program Replication Period), with noticeable differences between some of the runs. Plots for some of these measures are presented for two example runs (10 and 17) in Figure 6.1.

Ideally, we would like to know whether the differences in these measures between any of the runs are statistically significant. Such differences would indicate that evolution might genuinely be treading a different path, for no other reason than the different seed used for the random number generator when the runs commenced. The choice of a statistical test for this task was not immediately obvious. We wished to avoid parametric tests, as we did not want to make assumptions about the population parameters

[3] In the following, the pairs of run results displayed in Figure 6.1 and in Figure 6.8 were generally chosen because they illustrate noticeably different results.

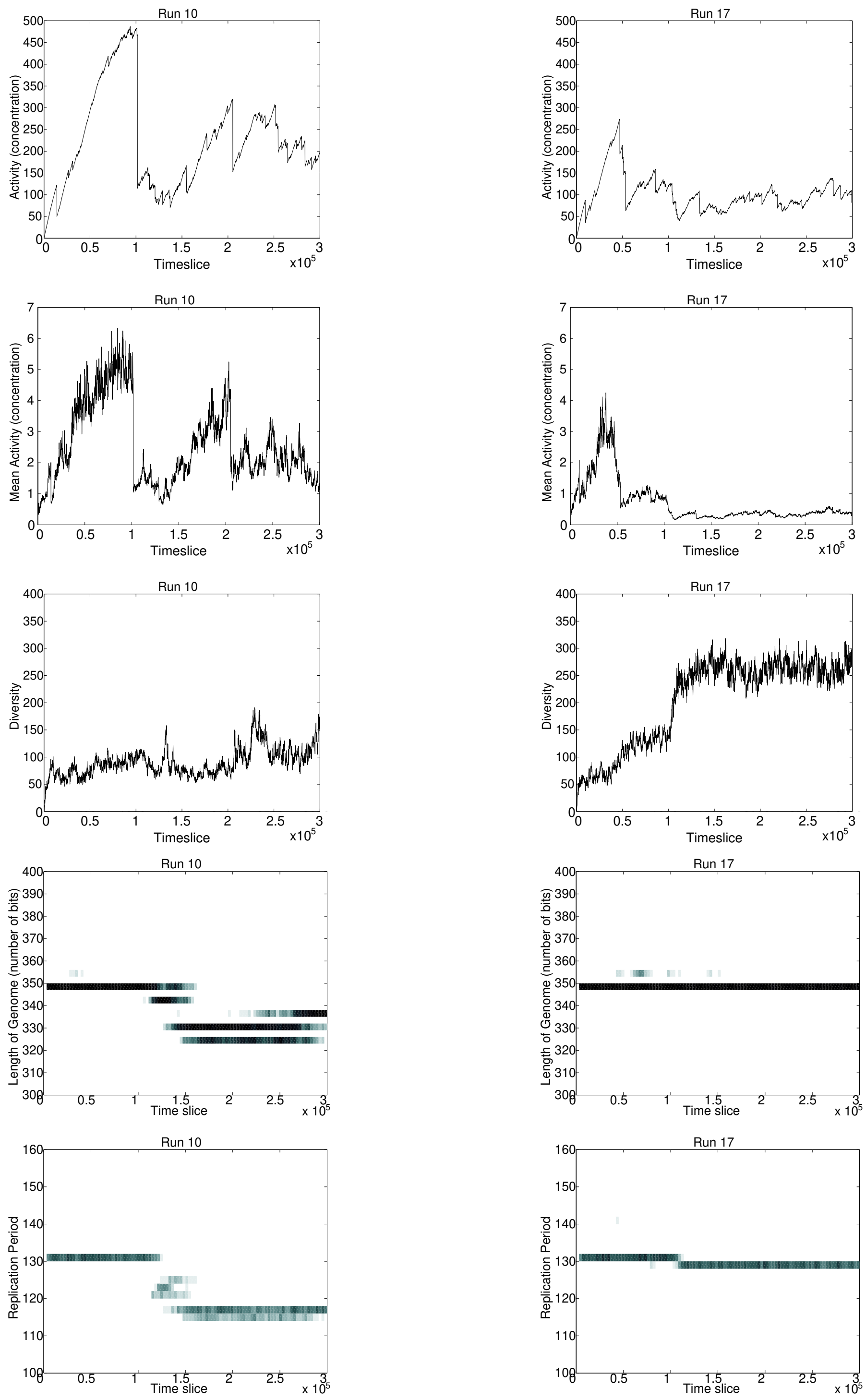

Figure 6.1: Contingency Experiments. Graphs of Cumulative Activity, Mean Cumulative Activity, Diversity, Program Length and Replication Period for Runs 10 (*left*) and 17 (*right*).

(for example, there is no reason to suspect that any of the measures we are looking at are normally distributed across all possible evolutionary runs).

We therefore chose a non-parametric method—a randomisation version of the paired sample t test (described in [Cohen 95]). For each measure of interest, this test will tell us, for each run, which other runs produced significantly different results. The test can indicate whether two samples are related without any reference to population parameters. The procedure used was as follows:

Procedure: Randomisation Version of the Paired-Sample t Test

For each run, 10 sample data points were extracted, each one representing the value of the measure in question at one of 10 equally spaced times throughout the run. The basic idea of the paired sample t test in this situation is to consider the 10 sample points for pairs of runs in turn. By doing pairwise tests at 10 sample points we are comparing the measures at a number of points through the run, with no point having more significance than any other. For each pair of runs, the difference between corresponding samples is calculated, together with the mean value for the 10 differences. We then ask what the likelihood is of achieving this mean difference under the null hypothesis that the two runs are statistically equivalent. The method by which this is done will be explained shortly.

Obtaining Raw Sample Points. In the case of measures which are already statistics of the whole population at any given time (i.e. Cumulative Activity, Mean Cumulative Activity, and Diversity), these 10 sample points could be taken directly from the value of the measure at the appropriate time. However, to prevent high-frequency changes in these measures from producing aberrant results, the measures were smoothed before the samples were taken (using median-smoothing with a window of 10,000 time slices).

In the case of the measures where the existing data consisted of multiple values at each time slice, each representing individual programs (i.e. the Program Length and Replication Period measures), each of the 10 sample points was produced by taking the median value of all values lying within a window of 1000 time slices around the time slice being sampled.

Obtaining Differenced Sample Points. Because of the cumulative nature of evolution, it is possible that a small difference in the sampled value of a measure early on in a pair of runs will be magnified into a large difference later on, even if the two runs are actually proceeding in a fairly similar fashion. In order to gauge the magnitude of this effect, a duplicate set of tests was run, which used the *difference* in value between adjacent sample points as the figure to compare between runs, rather than the *absolute* value of the sample points. Using differenced data should reduce the influence of any cumulative disparity between runs.

Testing for Significance. We are considering the difference in values between corresponding sample points in a pair of runs. Under the null hypothesis that the two runs are equal, however, it is equally likely that these values would be reversed (i.e. for sample point n for runs A and B, the null hypothesis is that the values A_n from run A and B_n from run B are just as likely to have come from the other run—A_n from run B and B_n from run A). If this were the case, the difference between the values would be the same as before, but with the sign reversed. We can test for the significance of the observed mean difference by constructing the distribution of all mean differences obtained from looking at each possible combination of each of the paired samples into one or other of the runs. As there are 10 paired samples, there are 2^{10} (1024) such combinations.

The exact procedure, which is adapted from [Cohen 95], is shown in Figure 6.2. The number that the procedure produces, p, is the (one-tailed) probability of achieving a result greater than or equal to $\bar{x}_D$ (or less than or equal to $\bar{x}_D$ if $\bar{x}_D < 0$) by chance under the null hypothesis. That is, p is the probability of incorrectly rejecting the null hypothesis that systems I and J have equal population mean scores for the measure in question.

For each of the five measures being considered (Cumulative Activity, Mean Cumulative Activity, Diversity, Program Length and Replication Period), this procedure was followed for each of the $19(19-1)/2 = 171$ pairwise comparisons between runs, for both the raw sample data and the differenced sample data.

The p values for each pairwise comparison are shown graphically in Figures 6.3–6.7. These figures show one histogram for p values obtained using raw sample data, and

1. For run I and J, if S_I and S_J are lists of the 10 sample data points for each run, construct a list D of the differences between these values, $D = S_I - S_J$. Denote the mean of these differences $\bar{x}_D$.
2. *if* $\bar{x}_D = 0$

   $p = 0.5$

   else

   (a) Set a counter C to zero.

   (b) for $i = 0..1023$
   - Construct a list D^* such that $D^*_j = D_j$ if $b_{ij} = 0$, or $D^*_j = -D_j$ if $b_{ij} = 1$, for $j = 1..10$, where b_{ij} is the j^{th} digit of i in base 2.
   - denote the mean of the new list $\bar{x}_{D^*}$
   - *if* $\bar{x}_D > 0$

     if $\bar{x}_{D^*} \geq \bar{x}_D$, then increment C by one

     else if $\bar{x}_D < 0$

     if $\bar{x}_{D^*} \leq \bar{x}_D$, then increment C by one

     endif

   (c) $p = (C/1024)$

Figure 6.2: Procedure for Significance Testing in Contingency Experiments.

another for p values obtained using differenced sample data. In all of the histograms, any p value less than 0.05 is plotted as zero. Bars of non-zero height on the histograms therefore represent pairs of runs which are not significantly different from each other for the measure in question at the $p = 0.05$ level.

(Note that, in order to emphasise the formation of various clusters of runs in these histograms, the runs in each histogram are arranged along the x and y axes in increasing order according to the mean of their 10 sample values. While this emphasises clusters in any one histogram, it means that clusters occurring in similar positions in the histograms of different measures do not necessarily represent the same runs.)

The randomisation version of the paired-sampled t test has some advantages over other methods of investigating pairwise comparisons (e.g. it is non-parametric), but it has the disadvantage that it is "virtually certain to produce some spurious pairwise comparisons" [Cohen 95] (p.203). Cohen suggests one way, not to get around this problem, but at least to have some idea of the reliability of a particular set of pairwise comparisons [Cohen 95] (p.204). The idea is to first calculate, at the 0.05 level, how many runs, on average, each run differed from (call this $\bar{n}_{0.05}$). Then calculate a similar figure at a much more stringent level. As we have 1024 numbers in our distribution of mean differences, the 0.001 level is appropriate. Finally, calculate the *criterion differential*, $C.D. = \bar{n}_{0.05} - \bar{n}_{0.001}$. If $C.D.$ is large, this indicates that many significant differences at the 0.05 level did not hold up at the 0.001 level. A small $C.D.$ value indicates that the experiment differentiates runs unequivocally, therefore lending more weight to the validity of the results at the 0.05 level. Table 6.2 shows $\bar{n}_{0.05}$, $\bar{n}_{0.001}$ and $C.D.$ for each measure, and for both raw and differenced sample data.

Table 6.2 reveals a number of interesting results. The most striking is the difference in the results of using raw sample points compared with differenced sample points.

Using raw data, the average number of runs that any particular run was significantly different to at the 0.05 level ranged from 8.42 for Cumulative Activity to 13.26 for Diversity. However, the criterion differential for all of these measures is high (ranging from 6.21 for Mean Cumulative Activity to 12.32 for Program Length). This suggests that the validity of the figures at the 0.05 level are questionable, and the true figures are probably somewhat lower than those calculated. Having said this, the average number of runs that any particular run was significantly different to, even at the 0.001 level, was non-zero for three of the measures (Cumulative Activity, 2.11; Mean Cumulative Activity, 4.11; Diversity, 6.32).

Using differenced data, the results have a very different look. In only two measures were any runs significantly different from any others even at the 0.05 level (0.11 for Cumulative Activity and 0.42 for Diversity), and both of these vanished at the 0.001 level. In other words, these figures suggest that, for *all* of these measures, starting off at any point during any of the runs, the amount the measure *changed* over a given period was not significantly different compared to any of the other runs.

6.1.3 Activity Wave Diagrams

Whereas the Activity and Mean Activity measures produce a summary figure for a whole population of genotypes at time t, activity wave diagrams plot the success of every genotype in the population at every stage of the run [Bedau & Brown 97]. They are therefore a useful visualisation technique for competition between genotypes, and the shape of an individual wave can also suggest the level of adaptive value of the

Measure	Data Type	$\bar{n}_{0.05}$	$\bar{n}_{0.001}$	$C.D.$
Cumulative Activity (concentration)	raw	8.42	2.11	6.32
	differenced	0.11	0.00	0.11
Mean Cumulative Activity (concentration)	raw	10.32	4.11	6.21
	differenced	0.00	0.00	0.00
Diversity	raw	13.26	6.32	6.95
	differenced	0.42	0.00	0.42
Program Length	raw	12.32	0.00	12.32
	differenced	0.00	0.00	0.00
Replication Period	raw	10.21	0.00	10.21
	differenced	0.00	0.00	0.00

Table 6.2: Mean Number of Runs that Each Run is Significantly Different from at the 0.05 Level ($\bar{n}_{0.05}$) and 0.001 Level ($\bar{n}_{0.001}$), and the Criterion Differential ($C.D.$). See text for details.

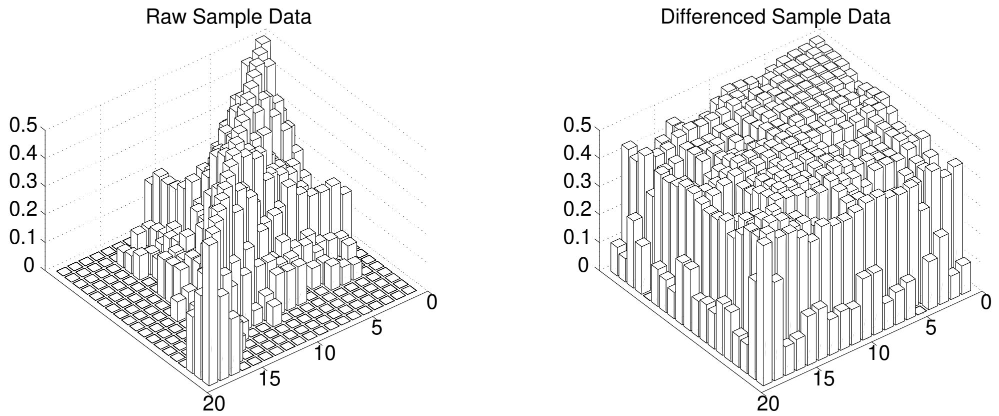

Figure 6.3: **Cumulative Activity (concentration)**: Pairwise comparisons (p values) between runs. Raw Sample Data (*left*). Differenced Sample Data (*right*). p values below 0.05 are plotted as zero, so bars of non-zero height indicate pairs of runs that are not significantly different at the 0.05 level. See text for details.

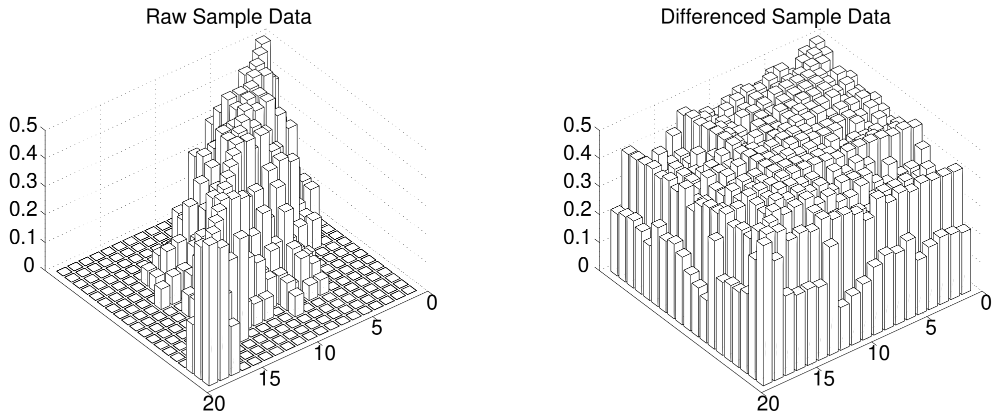

Figure 6.4: **Mean Cumulative Activity (concentration)**: Pairwise comparisons between runs. See text and caption of Figure 6.3 for details.

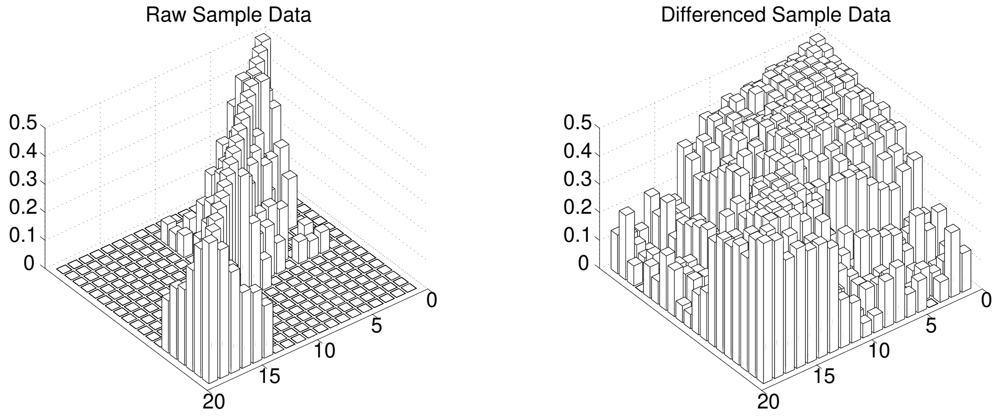

Figure 6.5: **Diversity**: Pairwise comparisons between runs. See text and caption of Figure 6.3 for details.

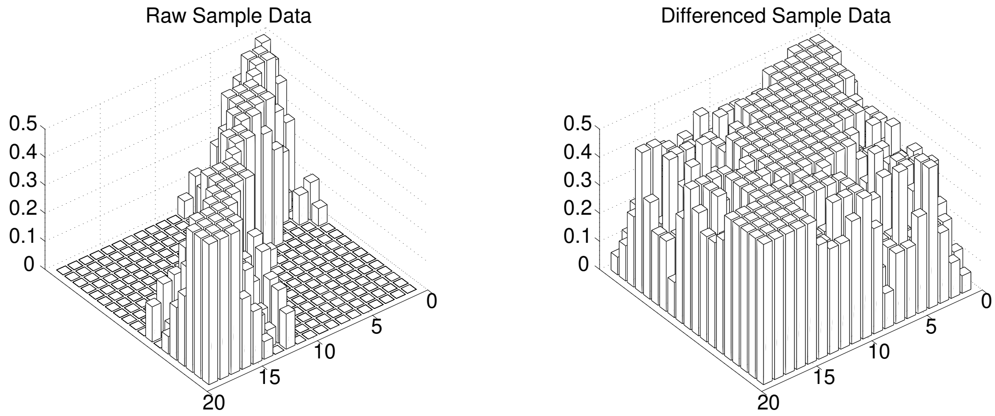

Figure 6.6: **Program Length**: Pairwise comparisons between runs. See text and caption of Figure 6.3 for details.

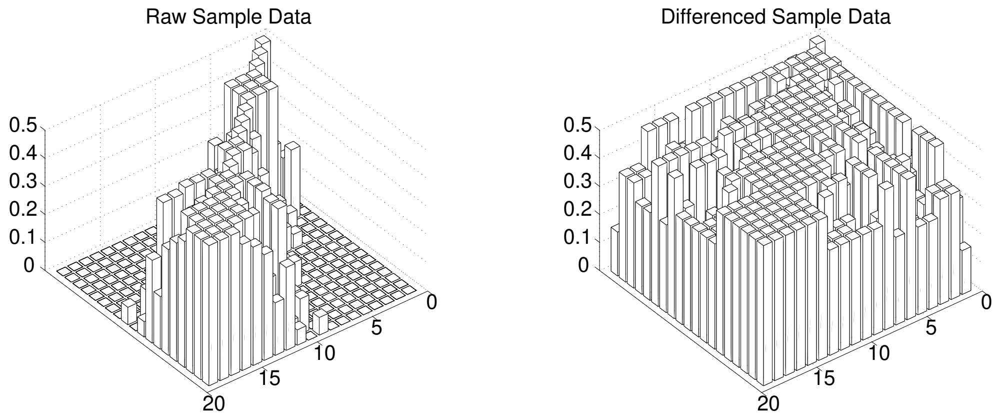

Figure 6.7: **Replication Period**: Pairwise comparisons between runs. See text and caption of Figure 6.3 for details.

corresponding genotype relative to its competitors (see Section 5.1.1).

The activity wave diagrams for most of the runs looked surprisingly different, although it is hard to quantify these differences (the Activity and Mean Activity measures do quantify some aspects of them, but no single measure captures all of the important information that the diagrams can tell us). Example activity wave diagrams (for runs 10 and 17) are presented in Figure 6.8.

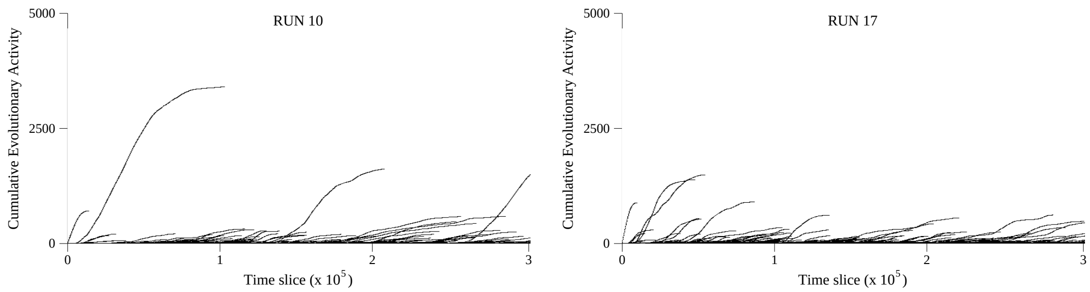

Figure 6.8: Activity Wave Diagram, Runs 10 (*left*) and 17 (*right*).

One way in which the activity wave diagrams can be very useful is in evaluating the effectiveness of different measures of evolution at highlighting the important adaptive events during a run. Cumulative Activity usually seems to give a better reflection of the wave diagram than does Mean Cumulative Activity. This is possibly because the latter measure is defined as Activity divided by Diversity, but diversity, by its very nature, does not take account of the *concentrations* of different genotypes, but merely their presence.

6.1.4 Discussion

Significant differences were found between runs for five of the measures investigated, at least when using the raw data. These measures were: Program Length, Replication Period, Cumulative Activity, Mean Cumulative Activity, and Diversity.

For Program Length and Replication Period, significant differences (at the 0.05 level) were observed in the raw data values between some runs. For these measures, the mean number of runs that each run is significantly different from at this level was calculated as 12.3 for Program Length and 10.2 for Replication Period, but the high criterion differential on these scores suggests that the true value should be somewhat lower (looking at Figures 6.6 and 6.7, probably somewhere in the range of 6 to 10).

Looking at the derived measures suggested by Bedau and colleagues (i.e. Cumulative Activity, Mean Cumulative Activity and Diversity), significant differences were found between runs which did hold up even at the 0.001 level. Again, the true value of each of these differences probably lay in the range of roughly 6 to 10.

These results indicate that each run, on average, performed significantly differently to between a third and a half of the other runs. One of the main reasons for doing these experiments was to understand how we should deal with contingency when conducting further experiments with Cosmos. If we assume that at least the finding that each run is statistically different to more than a third of the others is a general result, then we can use the following rule of thumb: For each re-run of a trial with a different seed for the RNG, the probability of its outcome being statistically equivalent (at the $p = 0.05$ level) to the original one is, at most, about $\frac{2}{3}$. Therefore, the number of *re*-runs that should be conducted to be confident (at the 95% level) of at least seeing one statistically different type of behaviour is n, where $(\frac{2}{3})^n \leq 0.05$, i.e. $n \geq 7.388$, or, in round figures, $n \geq 8$. This is the number of *re*-runs *after* the original, so, finally, we can say that *any trial should be conducted nine times with different seeds for the RNG.*

Having said that each run performed significantly differently to at least a third of the other runs, precisely *which* runs were significantly different depended upon the particular measure being looked at. This emphasises the fact that one should be clear about exactly what measure is being used when talking about comparisons between evolutionary runs.

The fact that *no* significant differences were found between any of the runs for any of the measures when looking at *differenced* sample data suggests that the significant differences observed in *raw* sample data may be caused (at least in part) by the cumulative magnification of initially small differences as a run proceeds. If this effect is controlled for (which was the purpose of using differenced data), the behaviour of the runs in terms of the *change* in values of the measures over a given time period would seem to be very similar in all of the runs. However, because of the cumulative magnification of small differences, the *absolute* outcomes of the runs *do* differ significantly in some cases, so contingency *does* play a big role.

As an aside, we can ask to what extent these results can be generalised to other evolutionary systems. Considering biological evolution first, it is clear that even just in terms of population size and the length of runs, the system is completely trivial. Also, the role of contingency may be different in systems which have rich ecological interactions (of which Cosmos programs have very little). It would therefore be unwise to claim that these results can tell us much about the role of contingency in biological evolution, but they may be relevant in specific cases. As for other artificial evolutionary systems, Cosmos is of comparable design, so the results, and the rule of thumb about the number of trials that should be run, should be broadly applicable to these platforms as well. The extent to which ecological interactions affect the results may be investigated by running similar trials on systems that display stronger interactions of this kind (such as Tierra).

In the remainder of this chapter, we will look at the effects of changing various parameter values on the behaviour of the system. In light of the results reported in this section, each of the following experiments is conducted nine times, using different seeds for the RNG each time.

6.2 Re-running the Standard Model

As the experiments reported in the previous section were done using slightly different parameter settings to those used for the standard run reported in Chapter 5, it was decided to re-run the standard run nine more times in order to see whether the results reported in that chapter were representative for the standard parameter settings. For each of the nine runs, all of the parameter settings were the same as those used in the previous chapter, except for the number used to seed the RNG at the start of the run.

6.2.1 Results

Some of the runs behaved very similarly to the one reported in Chapter 5, but others did not. By studying the results, it turns out that those runs which did behave similarly to the standard run were all examples of Bedau's Class 2 systems (see p.126), and those which behaved differently were all examples of Class 1 systems, as explained below.

Class 2 Systems

The results of four of the nine new runs (Runs 6–9) were very similar to those reported for the standard run. They had bounded Mean Cumulative Activity ($\bar{A}_{cum}$), positive Mean New Activity ($\bar{A}_{new}$), and bounded Diversity (D). At the end of each run, the diversity was very low, with a single dominant genotype and a dozen or fewer mutant genotypes present in much smaller numbers.

The results of Run 7 were slightly different because a mutant of length 372 bits arose which produced an offspring of length 378, which in turn produced an offspring of length 372, and so on in a cycle. However, the general pattern was the same as in Runs 6, 8 and 9.

Class 1 Systems

The results of the other five runs (Runs 1–5) were different. Each of these runs started off in a similar manner to those already described, but a some point during the run, their behaviour changed. The behaviour of the runs after this turning point was reached corresponds to Bedau's Class 1, with bounded D, zero $\bar{A}_{cum}$, and zero $\bar{A}_{new}$. The time at which this change occurred was different in each of the five runs, ranging from time slice 45,000 (approximately) in Run 2 to time slice 770,000 (approximately) in Run 4.

From the measures we looked at, reliable indicators of Class 1 dynamics were a combination of zero Mean Cumulative Activity ($\bar{A}_{cum}$), zero Mean New Activity ($\bar{A}_{new}$), stable population size, high diversity (of the same order as the population size), and high program length. Graphs of these measures for a typical run showing Class 1 behaviour are shown in Figure 6.9, alongside the corresponding graphs for a run showing Class 2 behaviour.

Analysis of these runs shows that in each one, at the time of the transition from Class 2 to Class 1 dynamics, a mutation appeared which caused a program not to stop copying instructions to its offspring after it had copied its final instruction, but rather to jump back and copy a section of itself a second time around. As a result, the offspring is approximately twice as long as its parent, but the first half of it is still a fully functional self-replication program. As long as the mutant retains the `stop` instruction at the end of the functional code, then the extra instructions do not get executed; they are the

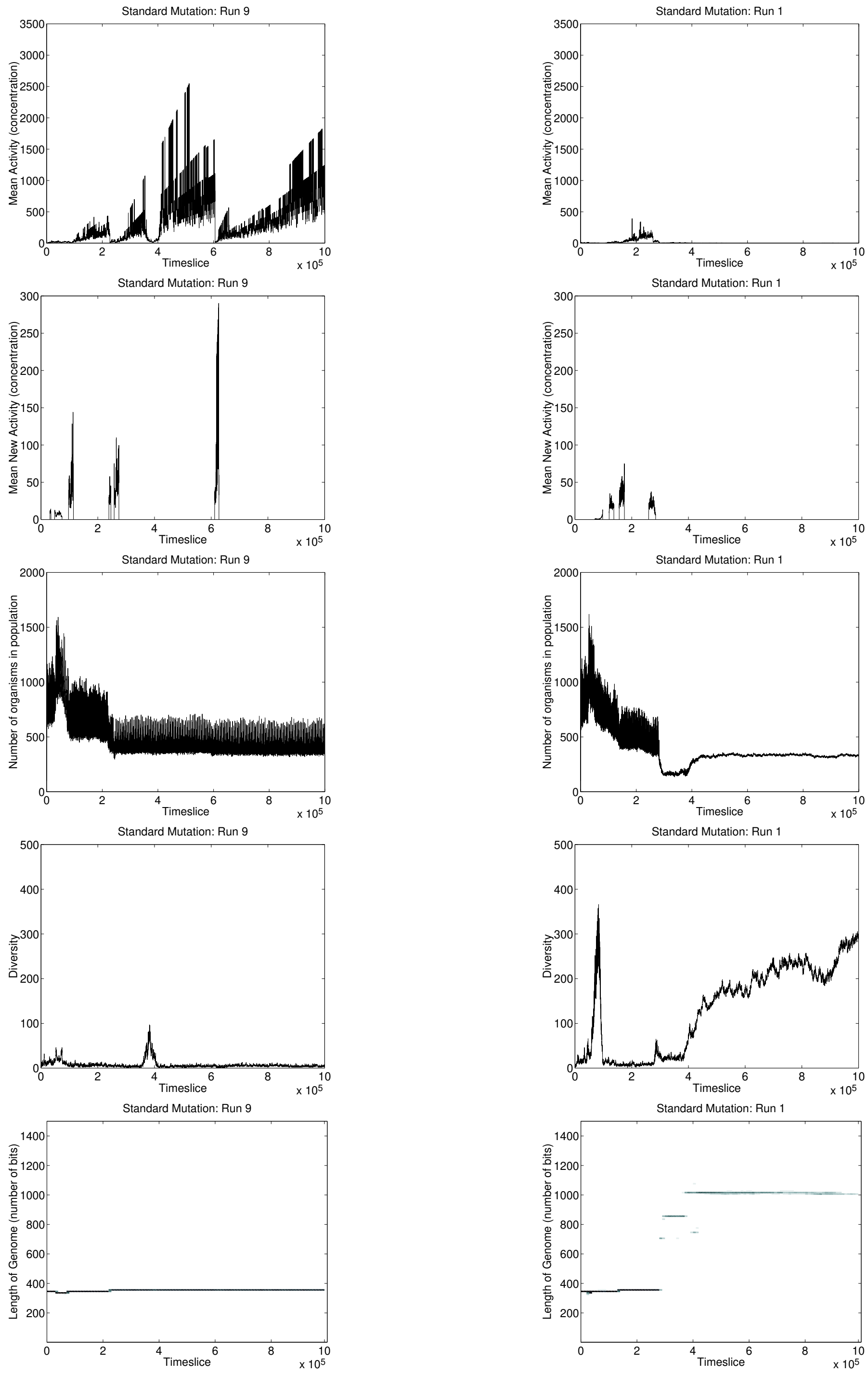

Figure 6.9: Class 2 versus Class 1 Dynamics. *Left*: Mean Cumulative Activity, Mean New Activity, Population Size, Diversity and Program Length for a Class 2 run (Run 9). *Right*: Corresponding graphs for a Class 1 run (Run 1, transition to Class 1 occurs at time slice $\sim 3 \times 10^5$).

equivalent of 'selfish DNA'. The Class 1 programs had also lost the ability to move around the environment.

Now, in Section 5.2.1 we looked at how the replication period of a program would be expected to vary as the program length increased under two different scenarios. In the first scenario, the length of the program's 'copy loop' increased in proportion to the total program length, and in the second scenario, it remained constant (Figure 5.8). In the present situation, the second scenario is the case. Under these conditions, we found that the replication period of the program would actually remain fairly constant no matter how long the program was. The longer programs in these runs are therefore not at any disadvantage to shorter ones in this respect, which explains why they were able to survive. However, to explain why these longer programs not only survived, but actually flourished and rapidly displaced the shorter programs, we need to look for specific selective *advantages* they might have had over the shorter programs. A likely explanation is that, because they are executing more instructions per time slice (due to their greater length), they collect more energy from the environment (by executing more `et_collect` instructions) during each time slice. In these runs, programs can extract energy from neighbouring programs as they try to collect energy from the environment (the parameter `energy_collection_scheme` is set to `shared`), so a program that extracts a large amount of energy from the environment per time slice might kill off some of its neighbours by using all of their energy. Those programs which have least stored energy, i.e. probably the shorter programs which collect less per time slice, will run a greater risk of being killed in this way, hence the longer programs will tend to out-compete them.

If this explanation is correct, then we would expect not to see this Class 1 behaviour if longer programs were not able to execute more instructions per time slice than shorter programs. We will look into this in Section 6.4.

Evidence to support this explanation comes from a number of different sources. First, if the long programs are extracting energy from their immediately neighbouring programs, then we might expect the system to move towards an equilibrium position where the eight locations neighbouring each viable program are empty (because any programs which happen to find themselves in these positions will quickly be killed by energy extraction). The largest number of programs that could be stably supported in this

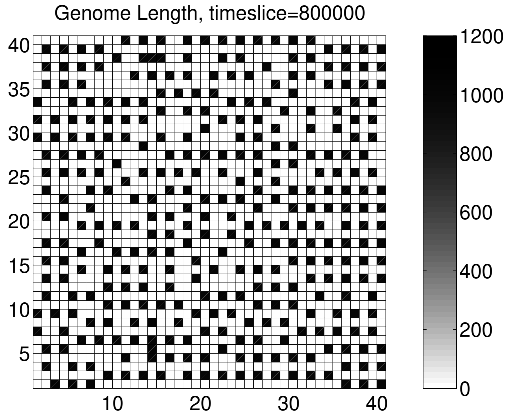

Figure 6.10: Standard Model Re-Runs: Spatial distribution of programs in Run 1 during Class 1 phase (time slice 800,000). Grey level represents program length. (White squares are empty.)

configuration would be, for the 40 x 40 grid used in these runs, 20 x 20 = 400 (i.e. alternate positions would be occupied and empty, in both directions). Looking at the population size graphs for each of the Class 1 runs, the population size does indeed move to around 400 at the time of the transition, and stay at that level thereafter. An example plot, for Run 1, is shown in the right-hand side of Figure 6.9. More direct support for the explanation comes from the recorded 'movies' of spatial distribution. A frame from the movie of program length for Run 1, at a time after the transition to Class 1 dynamics, is shown in Figure 6.10. This figure demonstrates that the spatial distribution is almost exactly as predicted (the observed pattern will never be exactly as predicted, because of the dynamic and stochastic nature of the system).

A consequence of the population size being limited to 400 is that it is no longer affected by the ceiling effect discussed in Section 5.2.3. The population size therefore becomes more stable, and is not subject to the oscillations observed in the Class 2 dynamics (compare the population size graphs for Class 1 versus Class 2 dynamics in Figure 6.9).

Another consequence of the spatial distribution is that the 400 or so programs which are favourably placed will tend to survive for a long time. This is because any new program that moves into a position in between two or more favourably placed programs will probably be killed off before it gets a chance to collect enough energy to fend off such attacks. The chances are weighted against the new program for two reasons: (i) being a new program, it probably has little stored energy to start with, and; (ii) because, even

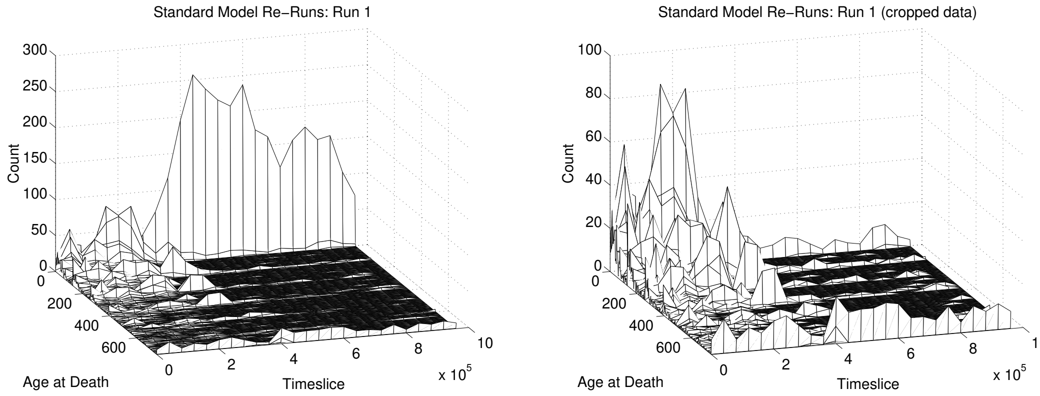

Figure 6.11: Standard Model Re-Runs: Age at Death of programs in Run 1. *Left*: All data. *Right*: Omitting data from programs that died within first 15 time slices after birth (i.e. first row of age axis). In these plots, the vertical axis ('count') shows the number of recorded organisms which died during that particular timeslice window that were of that particular age. The final row on the age axis show the number of programs that lived for *longer than* 750 timeslices.

though at each time slice the order in which programs are given a chance to execute instructions is determined at random, there will probably be two or four favourably placed programs surrounding the new, unfavourably placed one, so it is likely that at least one of these will be executed before the new program is. Evidence for this comes from the graphs of age-at-death. These are shown, for Run 1, in Figure 6.11 (plotted in 3D for clarity). The left-hand side of this figure includes all of the recorded data, and shows that after the transition to Class 1 dynamics at around 400,000 time slices, the number of programs that are dying within their first 15 time slices increases dramatically (these are the programs placed in unfavourable positions). On the right-hand side of the figure, the same data is shown with the first row (corresponding to programs which died in their first 15 time slices) omitted, to emphasise the rest of the data (note the difference in vertical scales). This plot shows that after the transition, programs tended to live for a very short time, or a very long time, with relatively few living for intermediate durations.

Finally, another consequence of the transition is that those programs which are favourably placed will tend to produce more offspring (because they live longer) than did most programs before the transition. This is demonstrated, for Run 1, in Figure 6.12. Note, however, that even though these programs are producing more offspring, most of these will be killed off immediately, as already explained. The fact that few programs will produce offspring which reach the reproductive stage themselves means that lineages

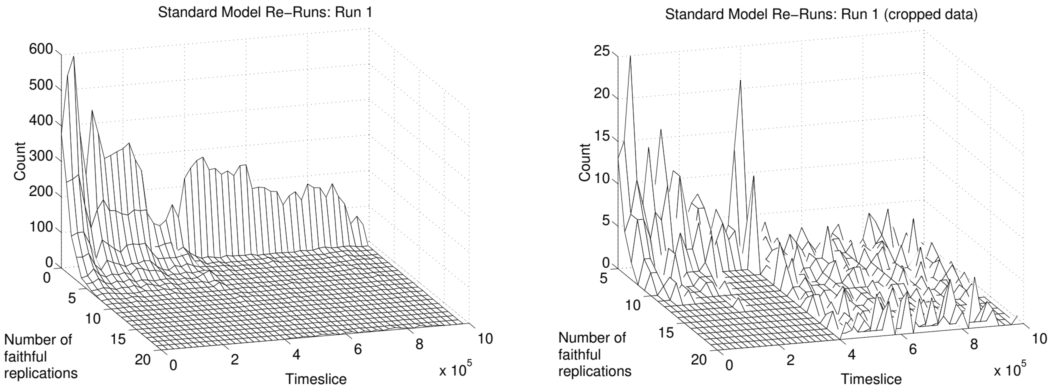

Figure 6.12: Standard Model Re-Runs: Number of faithful offspring of programs in Run 1. *Left*: All data. *Right*: Omitting data from programs that faithfully reproduced fewer than 6 times. In these plots, the vertical axis ('count') shows the number of recorded organisms which died during that particular timeslice window that faithfully reproduced exactly that number of times. The final row on the 'number of faithful replications' axis show the number of programs that faithfully replicated *more than* 20 times.

will not spread through the system, and explains why the diversity of the programs is very high during Class 1 dynamics. (The fact that there was presumably no selection pressure to prevent mutations in the non-functional (selfish) section of the programs must have also contributed to the high diversity.) Class 1 is therefore a degenerative state in which adaptations cannot spread throughout the population, and evolution effectively ceases.

6.3 Mutations and Flaws

In the standard run, the `mutation_period` parameter was set at 1,000,000, and the `default_flaw_period` parameter was set at 250,000. The behaviour of the system was investigated when these parameters were set an order of magnitude higher, and an order of magnitude lower.

6.3.1 High Mutation and Flaw Rates

The non-default parameter settings used for these experiments are shown in Table 6.3.

Parameter	*Value*
`mutation_period`	100000
`default_flaw_period`	25000

Table 6.3: Non-default Parameter Values for High Mutation Experiments.

Results

With high mutation and flaw rates, eight of the nine runs rapidly fell into the degenerative Class 1 dynamics. The transition occurred within the first 300,000 time slices in all of these cases. One of the runs, number 2, showed slightly different behaviour. The population size settled down to around 400 programs as is typical for Class 1 systems, but the diversity was somewhat lower than expected for Class 1 dynamics, it had non-zero mean cumulative activity and mean new activity, and the length of the programs was much shorter than is typical (around 400 bits, compared to the more usual 1000–1200 bits for Class 1 systems).

Analysis of the programs that lived in Run 2 after the initial transition from Class 2 dynamics revealed that they were almost identical to the ancestral program 348AAAA. The only major differences were that the programs had lost the ability to move (as is the case for normal Class 1 programs), but they also had in the order of *ten* extra `et_collect` instructions in their copy loops, in addition to the two which were present in the ancestor. Now, the addition of these energy collection instructions to the copy loop is the standard method by which the programs improve their ability to survive, as we saw in the analysis of the standard run in Section 5.2.7. However, in that analysis, we only saw programs which had gained an additional *seven* `et_collect` instructions in their copy loops.

It could be that a threshold exists at a certain number of extra `et_collect`s, after which point the programs are extracting too much energy from their neighbours to allow them all to coexist, thereby bringing on the Class 1 dynamics. This scenario would just be a slightly different method of bringing about the same result: in the Class 1 systems we have seen before, the programs execute more instructions per time slice because of their length, and therefore can run through several iterations of the copy loop and consequently execute many `et_collect` instructions per time slice; in the present case, the programs execute fewer instructions per time slice (because they are shorter), but still execute many `et_collect` instructions in a single iteration of the copy loop. To test this explanation, a number of runs were conducted using different ancestor programs for inoculation. Between runs, the ancestors differed in the number of `et_collect` instructions contained in their copy loops. Apart from this, they were the same as the standard ancestor 348AAAA, except that they all also had the `move` instruction removed so that they were sessile. All of the parameters took on their default values, except that mutations and flaws were switched off to enable us to study the neutral (non-adaptive) dynamics of the model. Each run lasted for 25,000 time slices. The population size graphs for four of these experiments are shown in Figure 6.13, for ancestors with 2, 5, 9 and 13 `et_collect` instructions inside the copy loop. These graphs clearly demonstrate that the dynamics do indeed start to change as the number of `et_collect`s rises, as predicted. It therefore seems reasonable to classify Run 2 as a Class 1 run, along with the other eight runs. In other words, all nine runs can therefore be classified as Class 1.

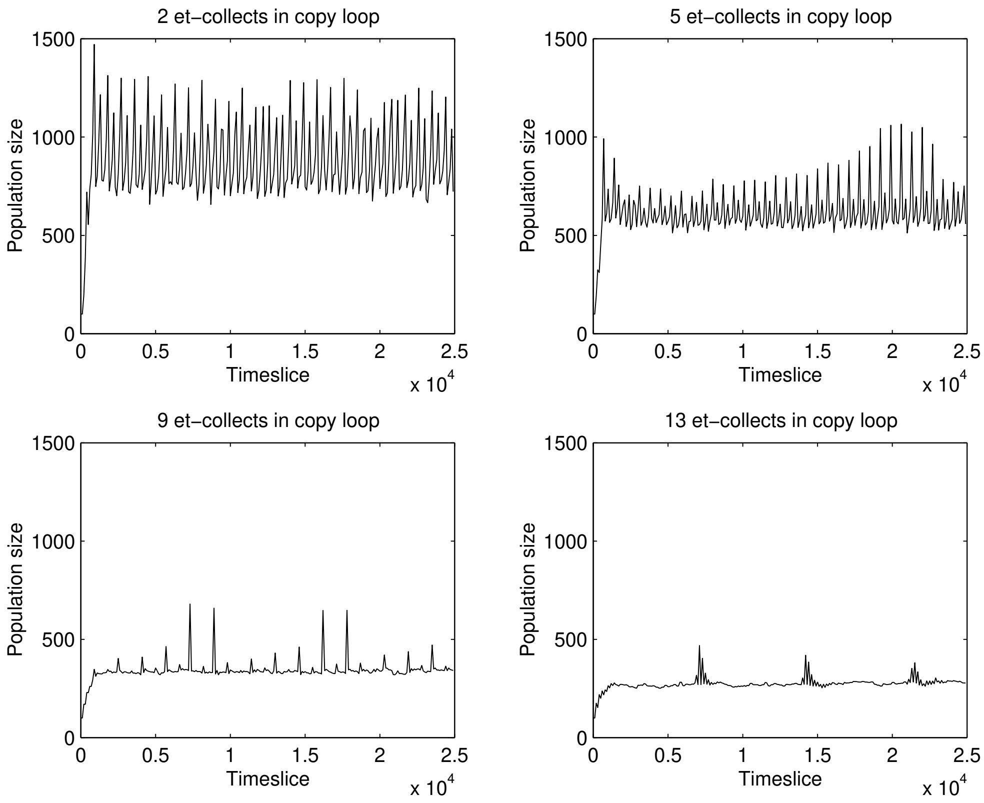

Figure 6.13: Population size graphs for ancestors with different numbers of `et_collect` instructions within copy loop. See text for details.

6.3.2 Low Mutation and Flaw Rates

Experiments were also conducted in which the mutation and flaw rates were much lower than in the standard run. The non-default parameter settings used for these experiments are shown in Table 6.4.

Parameter	*Value*
mutation_period	10000000
default_flaw_period	2500000

Table 6.4: Non-default Parameter Values for Low Mutation Experiments.

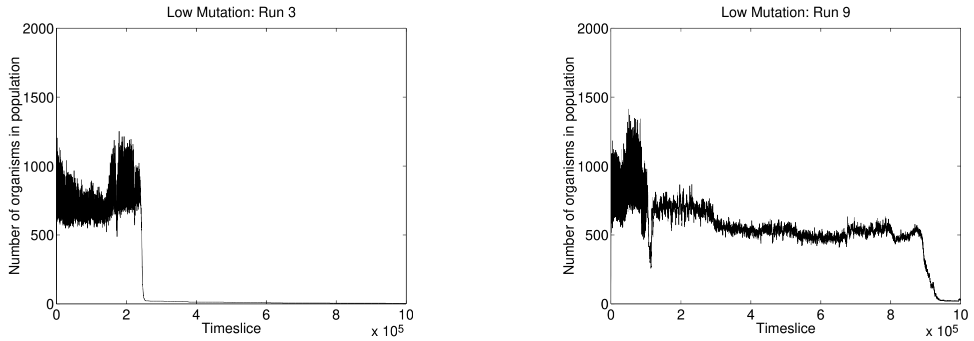

Figure 6.14: Low Mutation: Population size graphs for Run 3 (*left*) and Run 9 (*right*). Both runs experience population extinctions.

Results

Two of the runs (numbers 2 and 8) displayed typical Class 2 dynamics for their whole duration, and four others (numbers 1, 5, 6 and 7) began as Class 2 then switched to Class 1 behaviour at some point during the run. As expected with the lower mutation and flaw rates, the timing of these transitions was generally later than in the runs reported previously, ranging from time slice 200,000 (approximately) in Run 1 to 750,000 (approximately) in Run 6.

The other three runs (numbers 3, 4 and 9) displayed behaviour which had not been previously encountered. Each of these three runs initially displayed Class 2 behaviour as normal. Run 9 subsequently switched to Class 1 behaviour. However, at some later point during each run, the entire population was suddenly wiped out. (In Runs 3 and 9 a handful of programs remained, but these had lost the ability to reproduce and therefore spent the rest of the run collecting energy from the environment with no competition.) The population size graphs for runs 3 and 9 are shown in Figure 6.14.

Analysis of Run 3 revealed that the population was being invaded by a sort of 'cancer' mutant. The effect of this mutant was to add an extra (non-functional) instruction to the end of the program each time it reproduced. The length of the program therefore increased by one instruction (6 bits) each generation. The original mutant (342ABQQ) had actually lost an instruction (`if_not_fl`) from within the copy loop, which meant that it could reproduce quicker than the standard programs. This gave it an immediate benefit and enabled it to quickly spread and displace the existing programs. However, once established in the system, the mutant's offspring gradually grew longer and longer. The additional code at the end of the programs was executed, but had no significant action, and in particular, it did not include any additional `et_collect` instructions. The only effect of this growth was therefore to drain the program's energy store. Once the growth passed a certain size, it could *completely* drain the program's energy, thereby killing it. The takeover of the population by this mutant can be seen for Run 3 in the sequence of plots shown in Figure 6.15. This is a good example of the 'short-sightedness' of evolution, where a variant with a short-term advantage is selected for, even though it eventually leads to the total collapse of the entire population.

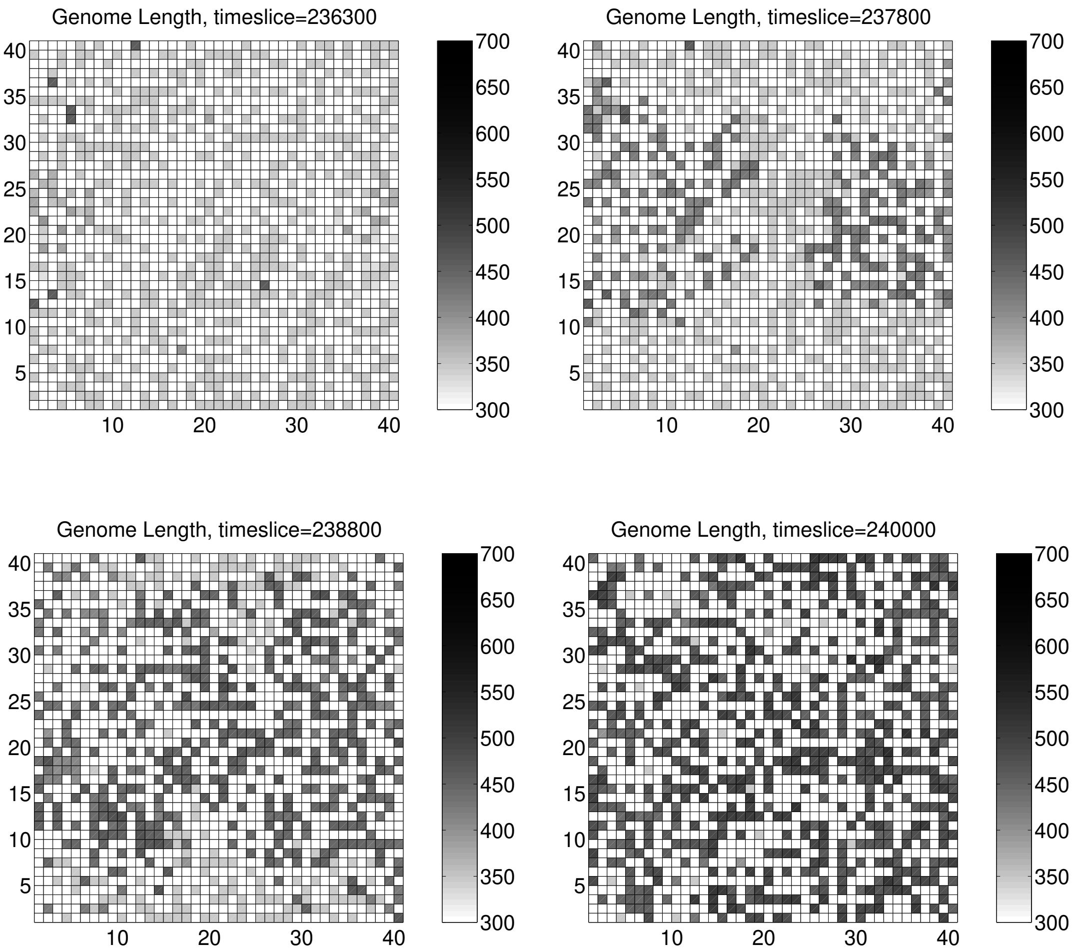

Figure 6.15: Low Mutation: Takeover of population by 'cancerous' mutant (darker squares) in Run 3. The sequence runs from top left to top right, bottom left then bottom right. The cancerous mutants can first be seen clearly as two clusters to the left and right of the central area of the top right figure. (White squares are empty.)

It is interesting that this mutant was only observed in these runs with low mutation and flaw rates. It is possible that such mutants did in fact arise in other runs, but with higher mutation and flaw rates, it is more likely that, once the detrimental effects of the mutant began to manifest themselves, a new variant would arise which corrects the defect. Such a variant would quickly displace the 'cancerous' programs which have low energy reserves, and thereby save the population from extinction.

6.4 The CPU-time Distribution Scheme

In the standard run reported in the previous chapter, CPU-time was distributed to the programs according to their length; longer programs were allowed to execute more instructions per time slice. Recall from Section 4.2.3 that the number of instructions, N, that a program of length L can execute at each time slice (provided it has enough stored energy tokens) is governed by the two parameters `et_value_constant` and `et_value_power` as follows:

$$N = \texttt{et\_value\_constant} * L^{\texttt{et\_value\_power}}$$

In the standard run, the values of these parameters were 0.025 and 1.0 respectively. In that run, the program length actually increased during the course of evolution, as described in Sections 5.2.1 and 5.2.7. To see whether this pattern could be repeated in the face of greater selective pressure for *shorter* programs, the run was repeated using the parameter values shown in Table 6.5. The values chosen were such that every program was allowed to execute 10 instructions per time slice, regardless of its length.

Parameter	*Value*
`et_value_constant`	`10.0`
`et_value_power`	`0.0`

Table 6.5: Non-default Parameter Values for CPU-time Distribution Experiments.

6.4.1 Results

All nine runs showed Class 2 dynamics for their whole duration. This therefore confirms the prediction made in Section 6.2.1 that Class 1 dynamics would not arise if longer programs had no advantage in terms of the number of instructions they were allowed to execute per time slice.