Evolving the Game of Life

Dimitar Kazakov[⋆]
[⋆]Department of Computer Science, University of York
Heslington, York YO10 5DD
kazakov@cs.york.ac.uk

Matthew Sweet[†]
[†]Department of Computer Science, University of York
Heslington, York YO10 5DD
mts104@cs.york.ac.uk

Abstract

It is difficult to define a set of rules for a cellular automaton (CA) such that creatures with life-like properties (stability and dynamic behaviour, reproducton and self-repair) can be grown from a large number of initial configurations. This work describes an evolutionary framework for the search of a CA with these properties. Instead of encoding them directly into the fitness function, we propose one, which maximises the variance of entropy across the CA grid. This fitness function promotes the existence of areas on the verge of chaos, where life is expected to thrive. The results are reported for the case of CA in which cells are in one of four possible states. We also describe a mechanism for fitness sharing that successfully speeds up the genetic search, both in terms of number of generations and CPU time.

1 Introduction

The aim of this work is to evolve a set (table) of rules for a cellular automaton (CA) with a good potential for producing "interesting" life forms, e.g., such that grow, move, have a long life span, consist of differentiated types of tissue, are compact and/or preserve their shape or produce a copy of themselves as a by-product of their growth. Rather than focussing on each such property in isolation, and trying to find the combination of rules that promotes it, an attempt is made to study an entropy-based criterion that is used as an indicator of the likelihood of such desirable life-forms appearing in a given CA. The criterion is used as a fitness function of a genetic algorithm (GA) searching through the range of possible CA. Also, to improve the GA performance on a task in which, intuitively, good solutions are few and far apart, we employ two techniques to maintain the balance between population quality and diversity — crowding (De Jong, 1975) and extended fitness — and show the substantial advantages that the latter brings in.

Cellular automata are dynamic systems consisting of a lattice of cells (in any number of dimensions) each of which has a number of associated discrete states k. The state of these cells is updated at discrete time steps, the resultant state dependent upon local state rules. Here we study two-dimensional automata with cells being in one of four possible states ($k = 4$, in other words, the cell belongs to one of three different types or is empty). The range of cells that influence the subsequent state of a cell is limited to immediate neighbours (sometimes denoted as $r = 1$ (Mitchell et al., 1993)). Each CA is defined by the table of rules that describe the subsequent state of the central cell for each 3×3 neighbourhood.

2 Genetic Algorithms and Extended Fitness

Genetic algorithms are search algorithms based on the mechanics of Darwinian evolution and genetics. GAs, despite their large variety, are all based on the same basic principles. They maintain a population of individuals representing candidate solutions to an optimisation problem. For each generation, a *fitness* reflecting estimated or actual quality is assigned to each solution (individual). A next generation of individuals is obtained by sampling the current so that individuals with higher fitness are favoured. Finally, the new generation is subjected to genetic operations such as crossover and mutation that aim at introducing a variety of new individuals. Then the cycle is repeated until some termination condition is fulfilled (Goldberg, 1989).

The fitness of an individual may measure the quality of the solution it proposes in absolute terms, say, as a scalar representing the value of a function that the GA is trying to maximise. In other cases, e.g., when it is normalised, fitness only represents the relative quality of an individual with respect to the rest. Fitness could be an even more abstract concept only reflecting the rank of the individual in the population.

Whether the goal of the GA is to provide a single best solution or a number of these, its principle remains the same: to store copies of the best one(s) aside from the main population, and wait until better ones are produced. For that ever to happen, it is essential that the GA should be capable of producing individuals that have not been seen in the previous generations. While the genetic operators, such as crossover and mutation, are the GA components that introduce new individuals in the population, their success depends on preserving sufficient genetic variety in it. Evolution, whether in nature or as implemented in GAs, can be seen as a dynamic process driven by two factors : (natural) *selection*, which favours the survival of the fittest, and *genetic variation*, which introduces new individuals, some of which could potentially outperform the best so far. Neither factor is sufficient on its own : without selection, the search for the best individual will become completely random ; without genetic variation, there will be nothing that selection can act upon as a uniform population of identical individuals reaches a dead end.

GAs employ several techniques that study the individual's fitness in the context of the whole population and modify it to preserve the balance between the forces of selection and those increasing genetic variation. Fitness *scaling* is used to reduce the risk of clones of one 'superindividual' taking over the whole population in the early stages of the search when the individuals' fitness is very varied and generally low. Also, in a population with a minimum of variety in the fitness, scaling helps emphasise the existing differences and promote the best individuals more strongly. In either case, scaling aims at normalising the differences between the fitness of individuals with respect to the extremes present in the population. This could be done in a number of ways, from using a linear scaling function to ranking individuals according to their fitness and substituting rank for the original fitness.

Holland (1975) has observed that for fitness functions with a rugged landscape two high fitness parents often generate 'lethals' (very low fitness offspring). De Jong (1975) suggests that this effect can be combated using an algorithm with crowding factoring such that a new individual will replace an individual from the previous generation with a similar genetic make up. For each child, a subset of the population is selected at random that contains k individuals and the member of that set closest (by bitwise comparison) to the new offspring is replaced. In this model, k is known as the crowding factor, and was shown by De Jong to have an optimal value of 2 over the complex multimodal foxhole function. In this work, trials have shown that 10 is an optimal value for the crowding factor given an initial population size of 150.

Inbreeding with intermittent crossbreeding is a technique proposed by Hollstien (1971) for search using genetic algorithms with multimodal fitness functions. The idea is that the individuals in each niche mate until the average niche fitness ceases to rise and then the individuals in that niche can mate with individuals in different niche. This allows the neighbourhood of each local maximum to be thoroughly searched before guiding the search for a global maximum to another unexplored part of the search space.

Although not used here, another technique worth mentioning is *niching*, which is used to modify fitness in order to avoid overpopulating parts of the search space in favour of others. In general, this technique is based on partitioning the range of all individuals into *niches* and

```
procedure evaluateExtendedFitness

for each chromosome C do
|extFitness(C) = 0
|  for each locus L do
|  |  extFitness(C,L) = 0
|  |  numberOfMatches=0
|  |  for each chromosome C′ do
|  |  |  if locus L in C = locus L in C′
|  |  |  increment numberOfMatches
|  |  |_ extFitness(C,L) += fitness(C′)
|  |  extFitness(C,L)/=(numberOfMatches *
|  |                        chromosomeLength )
|_ |_ extFitness(C) += extFitness(C,L)
```

Figure 1: Procedure computing extended fitness.

then reducing the fitness of individuals in overpopulated niches (Mahfoud, 1995).

Yet another alternative proposed here is the approach we call *extended fitness* (see Figure 1), in analogy to Dawkins's notion of extended genotype (Dawkins, 1982). As in population genetics, the components of this fitness are defined to measure the relative advantage that a gene gives to its carrier with respect to all other genes that can appear in the same locus. The fitness of the whole genome then can be computed as the averaged contribution of all its loci (Falconer, 1981). Extended fitness favours individuals with good genetic material (building blocks) and relies on the assumption that these could potentially be useful in the evolutionary search. Figure 2 shows the actual implementation of the way extended fitness is computed, which is faster, but more difficult to follow than the one shown in Figure 1. The overhead *O(numberOfChromosomes*chromosomeLength)* introduced by this latter implementation is very modest, and, as the experimental section of this article shows, it can be easily outweighed by the benefits it brings.

3 Evolving Cellular Automata

Using genetic algorithms for the search of CA with desired properties is a trend with a relatively short history. Previous research has often focussed on one-dimensional automata (Packard, 1988; Mitchell et al., 1993) or ones with two cell states ($k = 2$) (Packard, 1988; Mitchell et al., 1993; Sapin et al., 2003). In all cases, the resulting cellular automata (sets of rules) fall into one of four classes as defined by Wolfram (1983). Class 1 automata evolve after a finite number of time steps from almost all initial configurations to a single unique homogenous configuration in which all cells in the automaton have the same value. Class 2 automata generate simple disjoint structures dependent upon their initial configuration. The evolution of class 3 automata produces chaotic patterns

```
procedure evaluateExtendedFitness2

for each locus L do
for each allele A in L do
|   fitness(L,A) = 0
|_ presence(L,A) = 0

for each chromosome C do
for each locus L in C do
for each allele A in L do
|   fitness(L,A) += standardfitness(C)
|_ presence(L,A)++

for each locus L do
for each allele A do
|_ fitness(L,A) = fitness(L,A) /      \
     ( presence(A) * chromosomeLength )

for each chromosome C do
|   extFitness(C) = 0
|   for each locus L in C do
|   for the allele A in L do
|_ |_ extFitness(C) += fitness(L,A)
```

Figure 2: More efficient computation of extended fitness.

from almost all initial configurations; the statistical properties, after sufficient time steps, of the patterns produced by almost all initial configurations is the same. All other automata fall into a fourth class where for most initial configurations the automaton cells will become almost entirely unpopulated, however some stable live structures will develop which will persist indefinitely.

It would be unlikely that interesting creatures could exist in class 1 automata since after a finite time period all cells in the automaton would have the same value; therefore no interesting creatures could persist past this point. Class 2 automata merely generate simple disjoint structures (either periodic or static), which means that no reproducing or dynamic creatures could be created. Class 3 automata cannot support interesting creatures since the patterns produced are chaotic. Therefore the focus of this paper must be the fourth class of automata since they are most likely to be able to satisfy the criteria of supporting interesting creatures. The above speculations should be compared with Wolfram's suggestion that the fourth class of CA is of sufficient complexity to support universal computation Wolfram (1984).

When the fitness of a set of rules is to be determined then the cellular automaton that is represented by that set of rules needs to be run on some initial configuration of cells in the lattice. There are two techniques to consider that have been used in previous research to generate initial states — random generation and specific pattern generation. In random generation, some portion of the board is populated at random with live cells, and the density of live cells is dependent upon the probability of each cell being live. In specific pattern generation, a user defined pattern is created usually at the centre of an otherwise unpopulated lattice.

In their work, Basanta *et al.* (Basanta, 2003) used a static initial state — only the central cell is live and all others are initially unoccupied. This reduces the amount of computation necessary for each fitness calculation. For an algorithm using randomised initial states, several runs are needed to attain an accurate fitness for the rule set, however with a single static initial state the fitness must only be calculated once.

The approach adopted here is based on two GAs. Each individual of the first GA encodes the rules of one CA. For each of these CA, another, nested, GA is used to search for an initial configuration of the given CA that has the highest possible fitness. The fitness of an initial configuration is computed by running the CA through a number of time steps, summing the fitness for each ot them. This fitness is then used as the fitness of the CA. The inner genetic algorithm uses the fitness function described in the next section to discover initial configurations which favour the evolution of interesting life. All this can be summarised as follows:

1. Use a GA to select the best CA (set of rules).

2. To evaluate each CA, use another GA to select the best initial configuration (IC) and use its fitness as the CA fitness.

3. To evaluate each pair (CA,IC), run CA with IC for a predefined number of steps, measuring fitness at each step, summing it up, and returning the total. In other words, a CA, as defined by its set of rules, is only as fit as the fittest initial configuration that has been found for it.

The fitness landscape for this problem is highly rugged and therefore one must consider techniques for improving the effectiveness of the genetic algorithm under these conditions, with a particular attention to fitness scaling. In this work, crowding and extended fitness have been employed and compared.

4 Entropy Based Fitness of Cellular Automata

In this section, we introduce the fitness criterion used by the inner of the two above mentioned GAs.

The entropy of a system is defined to be the level of orderliness or chaos in that system – the higher the level of chaos, the higher the entropy. Wolfram (1983) defines the entropy of a CA to be:

$$S = -\sum_i p_i \ \log_2 p_i \tag{1}$$

where S is the entropy and p_i is the probability of state i. For two-dimensional automata an equation was developed by Wolfram and Packard (1985) to express the *set entropy* of an automaton.

$$S = \lim_{X,Y\to\infty} \frac{1}{XY} \log_k N(X,Y) \qquad (2)$$

where S is the entropy, X and Y are dimensions of the area for which the entropy is being calculated and k is the number of different states. When used to calculate the entropy of a particular state, N(X,Y) is the number of possible different states by which the current configuration can be represented. E.g., a 3×3 area containing one live cell could be represented by 9 different states, so N(X, Y) = 9. The division by XY normalises the entropy values, so that entropies over different tile sizes can be compared.

To calculate an approximation of this set entropy is far cheaper than to calculate the entropy using the first equation, since this first option would require us to enumerate all possible states. Also we are not interested in the overall automaton entropy but rather in the entropy of *tiles* (Sapin et al., 2003), a lattice of cells which is a component of the overall lattice of cells making up the automaton.

Another possible method of calculating the entropy would be to use the site entropy approach used by Langton. This is based on the entropy of a single cell of the lattice, and is defined to be 1 if the cell is in a different state to its state in the previous time step. The entropy of a tile therefore would be the sum of the entropies of all its constituent cells. To normalise the tile entropies, as with the set entropy calculation, it would be necessary to divide the tile entropy by the tile size, which gives the proportion of the cells that are in the same state as in the previous time state.

If we based the fitness function on the overall entropy then selecting for high entropy would result in a chaotic class 3 automaton which could not support interesting life. Conversely, selecting for a high degree of order (low entropy) would favour class 1 and class 2 automata which reach a stable state and never leave it. Interesting life is most likely to develop on the boundary between order and chaos; we need dynamic behaviour inherent in chaotic systems but we also need a degree of order to keep any developing life coherent. Therefore we wish to promote rule sets that contain both order and chaos — this can be achieved by assigning a *fitness based on the spread (standard deviation) of the entropies of (a sample of) the tiles making up the automaton*. We have also hoped that setting $k = 4$ would allow for specialisation among the types of cell, in a way specialised types of cells (tissue) have evolved in nature.

5 Results and Evaluation

A rule set was generated using a 100×100 board, and an initial population of 150 rule sets for the outer entropy based GA, which was run for 300 generations. The inner genetic algorithm, evolving for a given rule set inital configurations that create interesting creatures, had a population of 10 and was run for 20 generations — the population size and number of generations have to be kept low since they have a large effect on the run time of the algorithm. To evaluate each initial configuration, the CA was run with the given set of rules and initial configuration for 200 steps. To compute the fitness of a configuration (CA board), only tiles of size 3×3 and 15×15 were considered.

The properties of the best rule set found have been empirically analysed and are as follows. No single cell of any colour can survive, and neither groups of cells which are of type 1 (red) or 2 (blue). Cells of type 3 (green) survive, unchanged and unproductive, in some compact formations of sufficient size. All other life seen so far under the rule set consists of more than one different type of cell (and this is observed, as a rule, whenever the experiment is repeated). We have produced a rule set which favours life consisting of different types of tissue, one of the aims of the work, without specifying this as part of the genetic algorithm.

Here are some examples of the ways different types of cell interact. Any cell of type 1 (red) requires another cell of the same type, as well as a cell of type 3 (blue) in its neighbourhood in order to survive. Red cells catalyse the growth of blue cells: any non-live cell adjacent to a red cell and a blue cell will grow into a live blue cell. As a result, a cluster of red and blue cells will gradually evolve into a connected core of red cells completely enclosed by the blue. Cells of type 3 (green) grow in the presence of other 4 of the same kind, as well as when a combination of 1 green and 1 red is present. Certain combinations of green and blue neighbours breed another blue cell, and often clusters of these two types of cell are stable or show periodic behaviour.

To summarise, one can see type 1 cells (red) as serving as an inert ‘skeleton’ which has to be protected by a layer of blue (type 2) cells; type 3 (green) cells promote periodic behaviour in combination with type 2, and can be grown themselves in the presence of red (type 1) cells. Figures 3–4 are examples of some of the small structures that survive in this CA. For configurations with sufficient density (e.g., 0.7) complex, connected structures such as the one in Figure 5 with stable and periodic components, spanning more than half the environment in each direction, emerge as a rule. Here size is an important factor. A large structure will show a periodic behaviour equal to the least common multiple of the periods of all dynamic substuctures. For substructures of period 3,4 and 5 (all observed), the overall period will be 60, etc.

Figure 6 shows the type of life produced by another rule set selected by our algorithm after 200 generations. This rule set started from almost any initial configuration will generate a pattern resembling that of animal spots or organs made of layers of tissue. The automaton is resilient

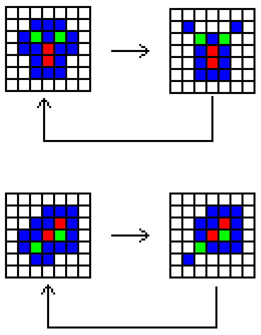

Figure 3: Cyclic structures with period 2.

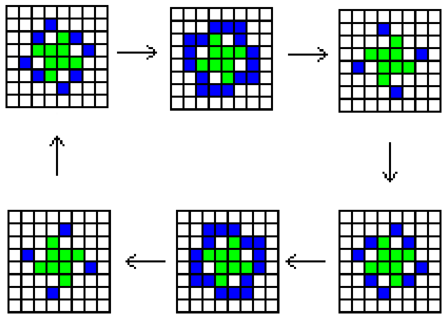

Figure 4: A cyclic structure with period 6.

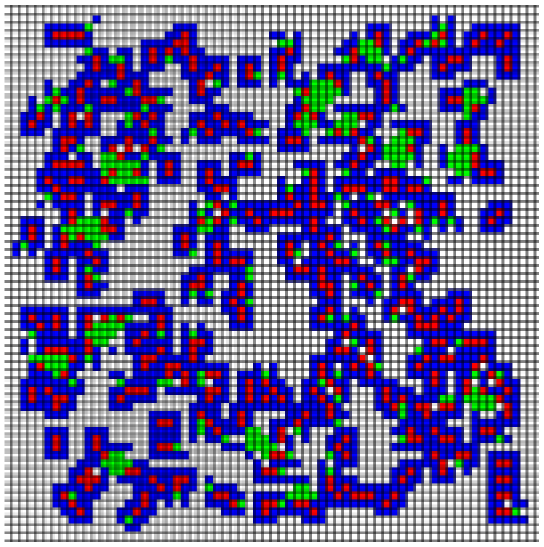

Figure 5: A large structure evolved from a dense, random configuration.

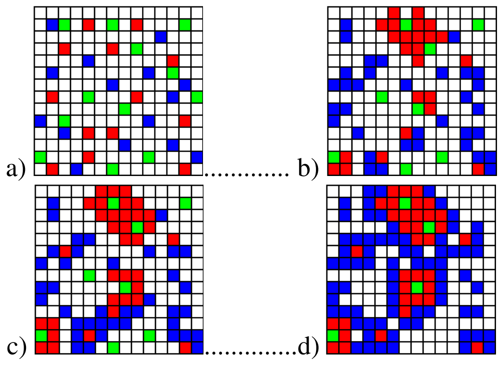

Figure 6: (a) Randomly generated initial configuration of the cellular automaton (density 0.2); (b) state after 1 transition; (c) state after 2 transitions; (d) state after a large number of transitions.

to damage and will regrow any killed cells so long as the core (green) cell is not removed.

Figures 7–8 show an interesting result related to the way GA is implemented: extended fitness produces faster improvements of the average population fitness despite the computational overhead it introduces. Moreover, extended fitness has a positive effect not only when the number of generations is compared, but also in terms of computational time, which is a very rewarding result. In a final observation on extended fitness (Figure 9), we have shown that the average population fitness can improve faster (again, in the stronger terms of time needed) with a larger population, which may seem counter-intuitive. This could be explained by the rugged fitness landscape which can be mapped more accurately by a larger population.

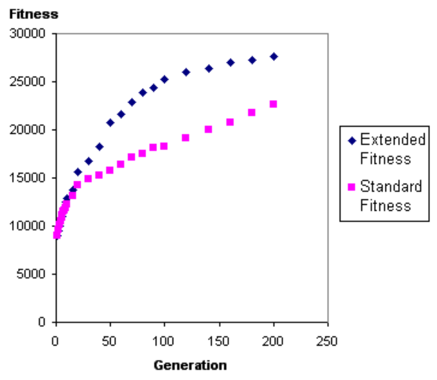

Figure 7: Comparison of the average fitness of rule sets for extended fitness and standard fitness approaches by generation (averaged over 10 runs using a population size of 100).

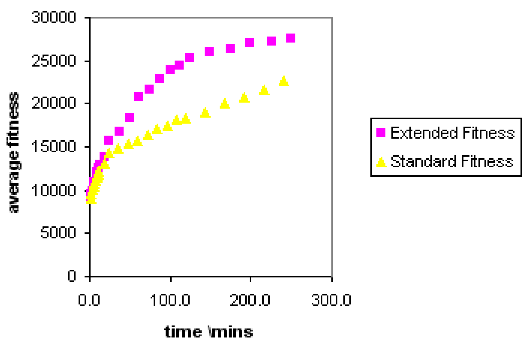

Figure 8: Comparison of the average fitness of rule sets for extended fitness and standard fitness approaches by time (averaged over 10 runs using a population size of 100).

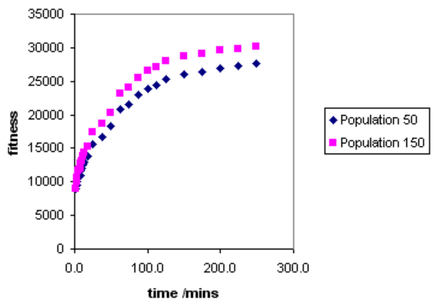

Figure 9: Comparison of the average fitness of rule sets for extended fitness and standard fitness approaches by population size (averaged over 10 runs).

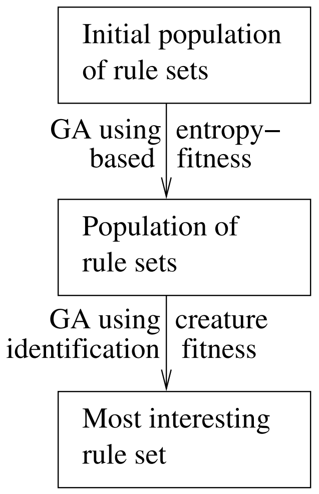
Initial population of rule sets
GA using entropy–based fitness
Population of rule sets
GA using creature identification fitness
Most interesting rule set

Figure 10: Generation of an interesting ruleset combining two different fitness measures.

6 Identifying Moving, Reproducing Creatures in Cellular Automata

In order to identify rule sets that support interesting life forms, the population of the genetic algorithm with the entropy based fitness function at the final generation will be used as the initial population to a second genetic algorithm. This second algorithm evaluates CA with a fitness function that identifies individual creatures (connected groups of cells) and maintains a table of those which it has seen. Creatures are identified by the proportion of each type of cell of which they are made up. The creatures which have been seen are stored in a list, the members of which are compared with the board at regular intervals. If a creature is seen again, then its past and present positions are compared. If the creature has moved, then the CA fitness is increased by the number of cells making up that creature. As a result, the fitness promotes CA generating long-living creatures that move. Creatures can be composed a minimum of 5 and a maximum of 200 live cells to reduce computation costs.

The problem with this fitness function is that it is very expensive to compute taking nearly 600ms (on the 700 MHz test machine). If we were to substitute this fitness function for the entropy-based one, and run the experiment described at the beginning of Section 5, it would mean calling it around 8.10^6 times, which would take approximately 8 weeks. Instead, the genetic algorithm using this fitness function was used to further evolve a population of rule sets generated by the an initial genetic algorithm using the entropy based fitness function (see Figure 10).

The process for identifying creatures is as follows:

1. Initially create a new array the same size as the game board, called the creature search array, and initialise all cells to ‘unchecked’.

2. Run through the game board sequentially checking each cell in the creature search array marked as ‘unchecked’.

3. When a cell in the game board is discovered which contains life then check the surrounding unchecked cells to identify if any of those are live. As each cell is checked remember that, so that it is not looked at again.

4. Keep a count of each type of cell encountered in the creature stopping once no more live cells are found to be connected to the original live cell. Now check that the creature size is between the minimum and maximum size for an organism; if so, keep a record of it, in terms of the three different cell types.

5. Continue traversing the game board until each cell is recorded as ‘checked’.

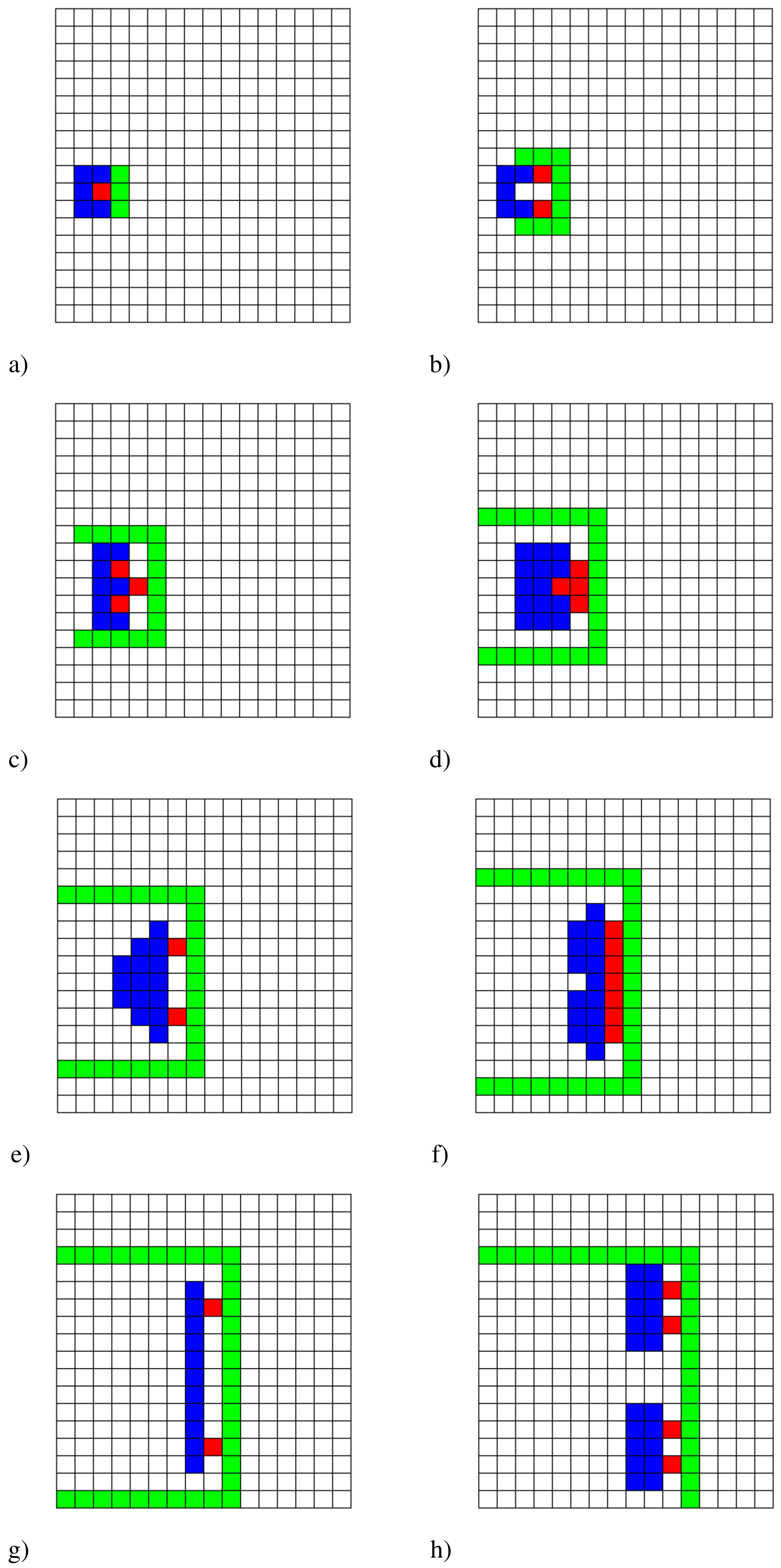

Figure 11: A trace of a creature over 8 consecutive steps. The creature moves to the right, expanding, and eventually splitting into two. Two very similar copies of the initial structure reappear in frame (g). In frame (h) they separate and the protruding 'nucleus' of each splits into two.

6. Now check to see if any of these creatures have been seen before since the start of this game: a creature with relative counts of each type of cell that fall within 5% of the counts of a previously seen creature, is seen as matching.

   do this by dividing the number of cells containing each life type by the number of cells containing that type for the recorded creature. Then compare the three values produced which should also be within a set percentage (5% during testing) of each other.

7. If a creature has not been seen before then it must be added to the list of creatures.

8. Check the creature has moved from its previous position before incrementing the fitness by the number of live cells in that creature. If more than one copy of a single type of creature is seen in a turn then multiply the number of live cells in all organisms of that type by the number of the organism present. For instance, if there are three copies of an eight-celled creature then each creature has a fitness of 24 and so a total of 72 is added to the turn fitness for all three creatures.

Using this genetic algorithm to further evolve the rule sets from the final generation of the entropy based genetic algorithm several interesting creatures were discovered. One of the most interesting is a moving reproducing creature. The creature is resistant to damage and splits every 11 time steps. Another example of moving, reproducing creatures is shown in Figure 11.

7 Discussion and Further Work

The experiments described above show that the fitness function based on entropy did, indeed, generate non-trivial cellular automata. The addition of another fitness function promoting moving and reproducing creatures has also achieved its goal. Future work should continue to concentate on automatically isolating and following the development of life-forms in a given CA. The behaviour of these creatures should be compared with the CA rules they are based on, in an attempt to find analogies with the processes of autocatalysis, mutual catalysis and inhibition, characteristic of biological systems.

Another achievement of this work is that it demonstrates the substantial benefits of using extended fitness on a task with an apparently very complex fitness landscape.

For the chosen type of CA (two-dimensional, $k = 4$ and $r = 1$), there are hundreds of rules in each rule set. Such large number of rules means it is difficult to analyse all the pathways in which cells can interact. One way to deal with this issue is to use a machine learning technique to summarise all rules and represent them in a more expressive formalism, stating, for instance, "cell of type X will be created if 2 to 4 cells of type Y are present", rather than enumerating these cases separately. Inductive Logic Programming (ILP) is an excellent candidate for the task Muggleton and Raedt (1994).

References

D. Basanta. Evolving automata to grow patterns. *Symposium on Evolvability and Interaction*, 2003.

R. Dawkins. *The Extended Phenotype*. Oxford University Press, 1982.

K.A. De Jong. *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, 1975.

D.S. Falconer. *Introduction to Quantitative Genetics*. Longman, London, second edition, 1981.

D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

J.H. Holland. *Adaption in natural and artificial systems*. University of Michigan Press, 1975.

R.B. Hollstien. *Artificial Genetic Adaption in Computer Control Systems*. PhD thesis, University of Michigan, 1971.

S. Mahfoud. *Niching Methods for Genetic Algorithms*. PhD thesis, University of Illinois, Urbana-Champaign, 1995.

Melanie Mitchell, Peter T. Hraber, and James P. Crutchfield. Revisiting the edge of chaos: Evolving cellular automata to perform computations. *Complex Systems*, 7:89–130, 1993.

S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19,20:629–679, 1994.

N. H. Packard. *Dynamic Patterns in Complex Systems*, chapter Adaptation towards the Edge of Chaos. World Scientific, 1988.

E. Sapin, O. Bailleux, and J. Chabrier. Research of complex forms in the cellular automata by evolutionary algorithms. In P. Liardet et al., editor, *Proc. of the* 6^{th} *Intl. Conf. on Artificial Evolution*, Marseille, Oct 2003.

S. Wolfram. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55, 1983.

S. Wolfram. Universality and complexity in cellular automata. *Physica D*, 10:1–35, 1984.

S. Wolfram and N. Packard. Two-dimensional cellular automata. *Statistical Physics*, 38:901–946, 1985.