A Cellular Genetic Algorithm Simulating Predator-Prey Interactions

Xiaodong Li and Stuart Sutherland

School of Computer Science and Information Technology
RMIT University GPO Box 2476v,
Melbourne, VIC 3001, Australia

ABSTRACT

This paper describes a cellular GA model using a new selection method inspired by predator-prey interactions. The model relies on the dynamics generated by spatial predator-prey interactions to maintain selection pressure and therefore population diversity. In this model, prey, which represent potential solutions, move around on a two-dimensional lattice and breed with other prey. The selection pressure is exerted by predators, which also roam around to keep the prey in check by removing the weakest prey in their vicinity. This kind of selection pressure efficiently drives the prey population to greater fitness over successive generations. Our preliminary study has shown that the predator-prey interaction dynamics play an important role in maintaining an appropriate selection pressure in the prey population, hence generating suitably fit prey solutions. Our experimental results are comparable or better in performance than those of a standard serial and distributed GA.

1. INTRODUCTION

Genetic algorithms are efficient algorithms for searching complex fitness landscapes by employing methods inspired by biological evolution. GAs have been successfully applied to many difficult optimization problems [1]. However, one common problem experienced when using GAs is premature convergence, where as a result of rapid loss of diversity in the GA population, the search is trapped to sub-optimal solutions. Many algorithms have been proposed to help maintain a more diverse population so as to prevent premature convergence. Among them, most popular are based on the idea of spatial separation [2,3,4]. In particular cellular GAs (or fine-grained Parallel GAs) have been shown to be very effective in maintaining population diversity [5,6,7]. In a cellular GA, individuals are commonly mapped onto a 2-dimensional lattice, with each cell corresponding to an individual. Selection and interaction (e.g., crossover) are restricted within the local neighbourhood of each individual. This "isolation-by-distance" feature allows a slow diffusion of good genes across the lattice, thereby maintaining a more diverse population than a serial GA with no such spatial structure. Most current work on CGAs use a static spatial structure that remains unchanged throughout a GA run. This contrasts with the fact that in nature we often observe dynamic spatial relationships among individuals.

Representation is another critical issue because GAs directly work on the coded representation of the problem. However, GA representations of solutions have been dominated by the use of fixed length and binary coded strings, largely due to the fact that in the early stage of GA research there was a specific emphasis on the use of binary representation [8,9]. The binary representation is often considered to be inappropriate in dealing with continuous search domains with large dimensions or when a higher numerical precision is required [9]. On the other hand, directly using a real-coded representation seems to be naturally suitable when dealing with problems using variables in a continuous domain. In this case, an individual is a vector of floating point numbers. These types of real-coded GAs have proven to be very successful [10,11,12].

In this research we aim to tackle the above issues of diversity and representation by extending a cellular GA in two ways. Firstly a selection method making use of the dynamics generated by predator-prey interactions is introduced to the GA population. We use prey individuals to represent solutions to the problem being optimized. These prey are allowed to wander around on a two-dimensional lattice and breed with their immediate neighbouring prey. There are also predators roaming around on the lattice that kill off the weakest prey, thereby encouraging fitter prey solutions to survive and breed. In contrast with the static spatial structure of a cellular GA, in this instance prey and predators are mobile, mimicking the dynamic spatial relationships observed in predator-prey interactions in nature. The essential aspect to the predator-prey interactions is that the predators keep the number of prey in check (i.e., to avoid prey population explosions) whilst not completely eliminating the prey population. One important feature that sets this model apart from other CGAs is that the selection pressure is maintained only through the predators killing off prey, rather than relying

on a direct replacement of the least fit individuals in the neighbourhood by the fitter offspring, which is what often occurs in a conventional CGA.

The second part of this work is to use real-coded GAs with suitable crossover and mutation operators. In conjunction with the use of dynamics of predator-prey interaction, we hope this extended model can outperform existing models.

The paper is organised as follows. Section 2 describes the proposed predator-prey CGA model, including the basic algorithm, a mechanism for balancing the predator and prey populations, and the selection and migration schemes. This is followed by section 3 describing the real-coded crossover and mutation operators. Section 4 describes the test functions used, the basic configuration for experiments and performance measurement criteria. Section 5 provides the experimental results as well as analysis. Finally, section 6 concludes with a summary of our findings and future research directions.

2. PREDATOR-PREY MODEL

The model consists of a two-dimensional lattice where the predator and prey populations reside. The lattice has its boundaries wrapped around to the opposite edge, therefore eliminating any boundary conditions. Each individual, whether predator or prey, is only allowed to occupy one cell at one time. At the beginning of a GA run, prey are randomly generated and distributed across the lattice. The predators, which keep the prey in check (i.e., their task is to kill the least fit prey), are also distributed randomly across the lattice. As illustrated in Figure 1, we normally start with a large number of prey and relatively small number of predators.

After the above initialisation, the predator-prey model proceeds in the following steps:

1) Each prey is given the chance to move into one of the neighbouring cells according to a pre-specified parameter *randomMoveProbability* (which is normally set to 0.5, so that half the prey would attempt to move one step on the lattice whereas the other half would remain where they were). If the prey were allowed to move, they could chose a random direction, i.e., one of the eight cells in a 8-cell Moore neighbourhood (north, south, east, west, and plus four diagonal neighbours), to move into. They then attempt to move. If the cells they are attempting to move into are occupied by other prey or predators, then they try again. This occurs 10 times. If the prey is still unable to find a place to move, it remains where it is.
2) After the prey have moved they are then allowed to breed. Each prey selects from its neighbours the fittest prey (excluding itself). If the prey has no neighbours it is not allowed to breed. Otherwise the prey and its fittest neighbour create an offspring using crossover and mutation operators (see section 3 for details). The creation of a new child is essentially one function evaluation. There are two methods for placing the child prey on the lattice. The first method randomly places the child on the lattice a maximum of two cells away from the parent prey. 10 attempts are tried to place the child, but if no free spot was found, the child is not generated. The second method simply places the child randomly anywhere on the lattice. Again 10 attempts are made to place it on the lattice. If all the attempted cells are occupied, the child is not generated.
3) The predators are then allowed their turn. The predators first look around their neighbourhood to see if there are any prey. If so, the predator selects the least-fit prey and kills it. The predator then moves onto the cell held by that prey. If a predator has no neighbouring prey, it moves in exactly the same way as prey (see above step 1). However it is possible to allow the predators to move more than once per prey time step (see section 2.1).
4) Go back to step 1), if the number of required evaluations is not reached.

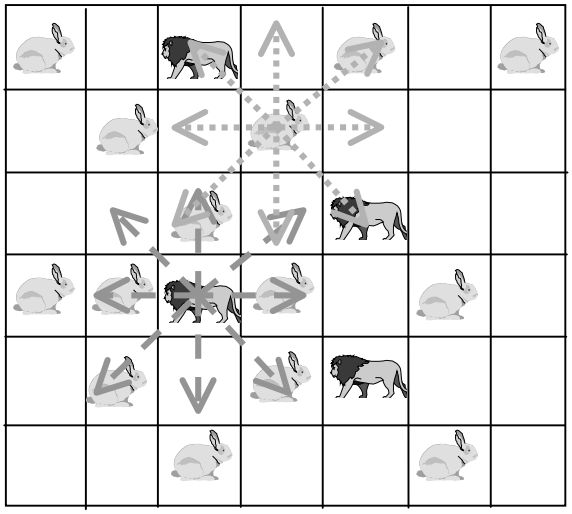


Figure 1: predators and prey are randomly distributed across the 2d lattice at the beginning of a run.

2.1 Balancing the predator and prey populations

One of the problems often encountered in predator-prey models is the difficulty of keeping a proper balance between the predator and prey populations. In order to prevent predators from completely wiping out the entire prey population, the following formula is adopted, where *iterations* is the number of moves the predators may take before the prey can make their moves:

$$iterations = \left\lfloor \frac{numPreys_{Actual} - numPreys_{Preferred}}{num_{Predators}} \right\rfloor \quad (1)$$

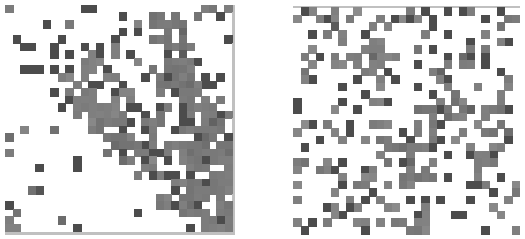
A predator can kill at most one prey per iteration, so Equation (1) is basically used to keep the actual number of prey ($numPrey_{Actual}$) to a number similar to the preferred number of prey ($numPrey_{Preferred}$). The predators are encouraged to minimise the difference between these two values. The floor operator ensures that the predators do not wipe out the prey population entirely. For example, if there are 450 prey, the preferred number of prey is 120, and the number of predators is 80, then the predators would iterate 4 times before the prey have a chance to move and breed again. Equation (1) is found to be useful in keeping a reasonably stable prey population size. Another merit of using equation (1) is that even when the minimum number of prey (the floor value) is reached, new born prey still get injected into the GA population, and they will stay alive and breed with other neighbouring prey until they get killed by the encountered predators. This can be seen as a mechanism of keeping a more diverse prey population.

2.2 Selection and migration methods

The elitist selection method is adopted in this model. Each prey is allowed to select the fittest prey from its eight neighbours (8 Moore neighbourhood) to breed with. The prey and its fittest neighbour create a child prey using real-coded crossover and mutation operators (see section 3).

After a child prey is created, we use two methods for placing the child on the lattice. These two methods are analogous to the migration schemes often used in an island model [3]. In the first method, the child prey is placed randomly up to two cells away from the parent prey. 10 attempts are tried to place the child, but if no free spot is found, then the child is not generated. The second method simply places the child randomly anywhere on the lattice. Again 10 attempts are made to place it on the lattice, but if no cells are free, then the child is not generated. The first method seems to mimic what occurs in nature, where prey tend to cluster together for protection, however the children prey tend only to find spaces at the edges of these clusters (see Figure 2 a).

Note that in this model we do not use any replacement scheme as in a typical CGA model, where often the offspring is used to replace the parent or the least-fit individual in its neighbourhood. The killing and removal of the least fit prey are only carried out by the roaming predators. Through this kind of predator-prey interaction, we hope an appropriate selection pressure may be maintained over the prey population.



a) b)

Figure 2: Predators and prey on a 2d lattice. Darker squares are the predators and lighter grey squares are the prey on the lattice. The variation in the greyness shows the fitness of prey. The brighter the square, the fitter a prey is. a) Prey tend to cluster together using the 1st selection method 1. b) Prey are not inclined to cluster using the 2nd selection method.

3. CROSSOVER AND MUTATION OPERATORS

We adopt a real-coded GA for the predator-prey model and test it over a number of benchmark test functions defined over a continuous domain. In our real-coded predator-prey GA model, each prey individual represents a chromosome that is a vector of genes, where each gene is a floating point number [10]. For example, a parameter vector corresponding to a GA individual can be represented as $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ ($x_i \in [a_i, b_i] \subset \Re$, $i = 1 \ldots n$). The GA works in exactly the same way as the binary counterpart except that the crossover and mutation operations are slightly different.

The real-coded crossover involves two operations. The first operation is a real crossover operator, which behaves similar to standard crossover. The difference is that instead of swapping binary values, the values in the slots of floating point array (i.e., a chromosome consisting of genes each representing a real-number variable) are swapped. For example, if we have two parents, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_n)$, and the crossover point is between x_i and x_{i+1}, then one child corresponds to $\mathbf{c1} = (x_1, x_2, \ldots, x_i, y_{i+1}, \ldots y_n)$ and the other $\mathbf{c2} = (y_1, y_2, \ldots, y_i, x_{i+1}, \ldots x_n)$. We apply this operator to 50% of the prey population.

The second crossover operator is so called blend crossover operator (BLX-α), first introduced by Eshelman and Schaffer [13]. BLX-α generates a child $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ where c_i is a randomly (and uniformly) chosen floating number from the interval $[\min_i - \Delta \cdot \alpha, \max_i + \Delta \cdot \alpha,]$, where $\max_i = \max\{x_i, y_i\}$, $\min_i = \min\{x_i, y_i\}$. Eshelman and Schaffer reported BLX-0.5 (with α=0.5) gives better results than BLX with other α values. It seems that with α=0.5, BLX provides a nice balance between exploration and exploitation (convergence), therefore we choose to use α=0.5 in this model.

We apply mutation with a probability to the entire prey population. It is a very simple mutation operator, which

Table 1: Summary of the results on the four test functions.

Config.	*Sphere*			*Griewangk*		
	A	***SD***	***B***	***A***	***SD***	***B***
nearby	1.70E-05	3.26E-05	2.24E-07	3.19E-02	2.21E-02	1.39E-03
lattice	2.79E-15	2.50E-15	3.63E-16	7.40E-04	2.26E-03	1.56E-13
R-BLX	9E-07	6E-07	2E-07	6E-01	1E-01	4E-01
D-BLX	1E-14	7E-15	3E00	2E-02	2E-02	4E-12
	Rastrigin			***Rosenbrock***		
nearby	2.30E-01	3.53E-01	5.53E-04	8.79E-03	1.21E-02	7.85E-07
lattice	2.53E-01	5.03E-01	2.51E-09	5.09E-03	9.91E-03	2.49E-07
R-BLX	4E01	9E00	2E01	3E01	3E01	2E01
D-BLX	1E01	3E00	7E00	2E01	1E01	2E01

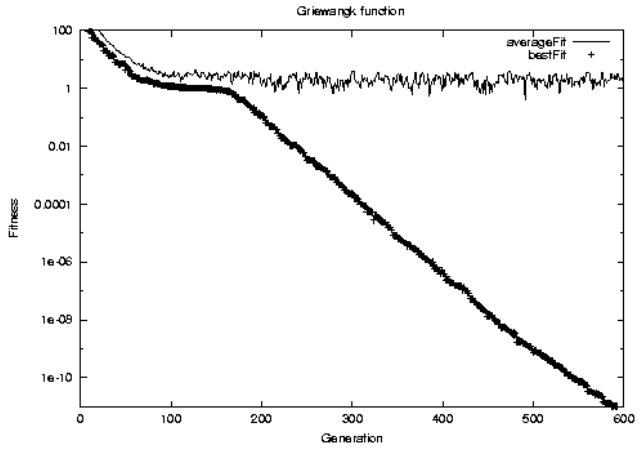


Figure 3. Average and best fitness of the prey population over a single run.

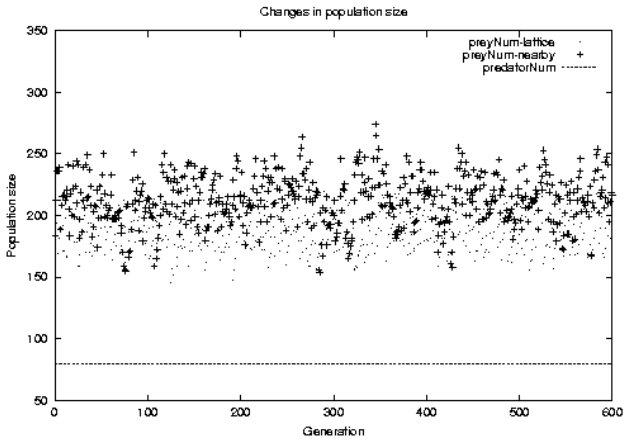


Figure 4. Variable prey population sizes and an invariant predator population size.

replaces a gene (i.e., a real parameter value) in a chromosome with another floating point number randomly chosen within the bounds of the parameter values.

4. EXPERIMENTAL DESIGN

We use a lattice size of 30x30, and apply to the 50% of the prey population a real-coded crossover and the remaining 50% a BLX-0.5 (α=0.5) crossover. The mutation probability is 0.01. We use the elitist selection in a local neighbourhood of 8 cells, and a *randomMoveProbability* of 0.5.

To evaluate the performance of the predator-prey model, four benchmark test functions, *Sphere*, *Griewangk*, *Generalized Rastrigin*, and *Generalized Rosenbrock*, are used, all of which have a global minimum of 0 [14]. The dimension *n* of these functions has been chosen to be 25. For each test function, we run the model 30 times (each time with a different random number seed), each run with 100,000 evaluations. We measure the performance of the model over the 30 runs based on the following criteria: ***A*** - the average of the best fitness found at the end of each run; ***SD*** - standard deviation at the end of each run; ***B*** - best of the fitness values found over the 30 runs. Two configurations have been tried, 'nearby', one with new born prey randomly distributed only within a distance of 2 cells from the parent, and 'lattice', which allows such random distribution over the entire lattice. We also measure the predator-prey model's ability of maintaining population diversity and the dynamic changes of the prey population size over generations. The results and analysis are given in the following section.

5. RESULTS AND ANALYSIS

Table 1 summarises the results of ***A***, ***SD***, and ***B*** using 'nearby' and 'lattice'. For comparison, we also include the results of a sequential GA using BLX-0.5 crossover operator (R-BLX) and a distributed GA using BLX-0.5 (D-BLX), as provide by Herrera and Lozana [14] (Table VI on p.53). These results, especially the results of the 'lattice' configuration compare favourably to those of R-BLX and D-BLX. Note that we tried to use the same performance measurement, and the dimension for the test functions (n=25) as those of Herrera and Lozana [14], in order to compare our results with theirs fairly. However,

their results on the R-BLX and D-BLX were obtained after 500,000 evaluations.

5.1 Maintaining population diversity

Figure 3 shows a typical single run of the predator-prey model on the *Griewangk* function with the 'lattice' option. Note that the x-axis shows the number of generations instead of evaluations, as at each generation, there are multiple new-born prey/evaluations. An evaluation is only carried out when a child prey is successfully produced. Although the average fitness of the prey population gets stagnant after generation 100, the best fitness continues to improve with no sign of slowing down. This is not surprising because of two reasons. Firstly the predators only kill prey within their vicinity (i.e., restricted killing), and secondly there is a constant influx of new-born prey into the prey population throughout the run. The new-born are not necessarily always the best-fit ones, therefore the average fitness value of the population remains relatively high (assuming minimization). This indicates that the overall diversity of the prey population is still rather high even when the model approaches the end of a run.

5.2 Prey and predator population sizes

Figure 4. shows the fluctuation of the prey population sizes in two settings, the 'lattice' and 'nearby', and the invariant number of predators in a single run. The use of `nearby' migration scheme shows that the clustering of prey (see Figure 2 a), to some extent, has an effect of preventing predators from killing off more prey in the population, as compared with the `lattice' setting (see Figure 2 b). Equation (1) seems to provide a very effective way in maintaining a variable but balanced prey population.

6. CONCLUSION

In this paper we have proposed the use of predator-prey interaction as an effective selection method in the context of a cellular GA model. Selection pressure is dynamically imposed upon the prey population through the killing off carried out by the roaming predators on a two-dimensional artificial world. Our experimental results have demonstrated that this type of selection method utilising the dynamics produced by predator-prey interactions is very effective in maintaining an appropriate selection pressure and diversity in the prey population, leading to a substantial improvement in performance. Compared with standard real-coded serial and distributed GAs, It has been shown empirically that the performance of the predator-prey CGA on all the four test functions are comparable or better than the existing models found in literature.

Future works will investigate the optimal predator-prey ratio and the effect of using different lattice sizes, or equivalently the population density of the prey and predators. We could also introduce additional biologically and ecologically inspired features such as an energy level and lifespan of an individual prey and predator, or food source as an environmental variable to the model. Extensions such as these would allow the predator-prey model to generate more complex dynamics analogous to our observation in nature, which are critical to the survival or demise of the prey and predator populations.

7. REFERENCES

[1] Goldberg, D. (1990), *Genetic Algorithms in Search, Optimisation and Machine Learning*, Addison-Wesley, Massachusetts.

[2] Tanese, R. (1989), Distributed Genetic Algorithms. In *Proceeding of the Third International Conference on Genetic Algorithms*, Schaffer, J.D. (Ed.), Morgan Kaufmann Publishers, San Mateo, p.434-439.

[3] Cantu-Paz, E. (1997), A Survey of Parallel Genetic Algorithms. *Technical Report IlliGAL 97003*, University of Illinois at Urbana-Champaign.

[4] Tomassini, M. (1999), Parallel and distributed evolutionary algorithms: a review, in *Evolutionary Algorithms in Engineering and Computer Science*, edited by Miettinen, K. *et al*., New York: John Wiley & Sons ltd, p.113-133.

[5] Manderick, B. and Spiessens, P. (1989), Fine-grained parallel genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, p. 428-433.

[6] Sarma, J. and De Jong, K. (1996), An analysis of the effects of neighborhood size and shape on local selecrion algorithms. In *Proc 4th PPSN, LNCS 1141*, Springer Verlag, p.236-244.

[7] Li, X. and Kirley, M. (2002), "The Effects of Varying Population Density in a Fine-grained Parallel Genetic Algorithm", in *Proceeding of 2002 Congress on Evolutionary Computation* (CEC), Honolulu, Hawaii, May 12-17, 2002.

[8] Goldberg, D.E. (1991). Real-Coded Genetic Algorithms, Virtual Alphabets, and Blocking. *Complex Systems* **5**, p.139 - 167.

[9] F. Herrera, M. Lozano, J.L. Verdegay (1998). Tackling Real-Coded Genetic Algorithms: Operators and tools for the Behaviour Analysis. *Artificial Intelligence Review* **12** (1998), p.265-319.

[10] Wright, A. (1991). Genetic Algorithms for Real Parameter Optimization. *Foundations of Genetic Algorithms 1*, G.J.E. Rawlin (Ed.), (Morgan Kaufmann, San Mateo), p.205 - 218.

[11] Davis, L. (1991). *Handbook of Genetic Algorithms.* Van Nostrand Reihold, New York.

[12] Muhlenbein, H. & Schlierkamp-Voosen, D. (1993). Predictive Models for the Breeder Genetic Algorithm I: Continuous Parameter Optimization. *Evolutionary Computation* **1**, p.25-49.

[13] Eshelman, L. J. and Schaffer, J. (1991). Realcoded genetic algorithms and interval-schemata. *Foundation of Genetic Algorithms*, p.187-202.

[14] F. Herrera, M. Lozano (2000). Gradual Distributed Real-Coded Genetic Algorithms. *IEEE Transactions on Evolutionary Computation* **4**:1 (2000), p.43-63.