Prototype Report

1 SN P Systems

A Spiking Neural P System (SN P) is a new computing model, originally proposed in 2006, inspired by the structure and functioning of neurons. These systems have the unique characteristic of using time as a computational unit, where each cycle *t* will cause every neuron to progress at the same time in a synchronised manner.

Each neuron contains a set of rules, which need to match exactly in order for a calculation to be performed. These calculations take form of either "spiking" or "forgetting". In the spiking scenario, a neuron will accept a rule defined as "$E/a^c \rightarrow a; d$ where E is a regular expression, over $\{a\}$ and c, d are natural numbers, c ≥1, d ≥ 0."[2] Meaning that a neuron which contains k amount of spikes, such that "$a^k \in L(E); k \geq c$, can consume *c* spikes and produce one spike, after a delay of *d* steps."[2]. This spike will be sent to all neurons that it has a connection to (a synapse), originating from the neuron in which the rule was applied. A formal definition of SN P systems can be found in [2] on pages 5 - 7.

In the forgetting scenario, the rules follow the same principle as when spiking, except they take the form of $a^s \rightarrow \lambda$, meaning that s ≥ spikes are to be removed, providing that the neuron contains exactly *s* amount of spikes. These spikes are then removed from the system.

An additional scenario is also considered in a generating mode of an SN P system, a mode which considers an initial spike configuration in the system and will generate an output, passed through to the environment. The neuron responsible for this is called the *output neuron*. There is another mode which SN P systems can take, which is an accepting configuration, where the output neurons can be omitted. For this project, only the generating SN P systems are considered.

There are many interesting implementations of SN P systems, such as networks which can perform logic, very much in the same way a Finite State Machine would. These examples can be found in [3].

Implementing this structure on a standard central processing unit (CPU) requires a slight adaptation in simulating SN P systems, as a true parallel solution cannot be created due to the sequential nature of CPUs. Instead, splitting the actions that a Neuron would perform and executing them sequentially on the same sets of information and then combining the outputs can give us an accurate representation of the synchronous nature of SN P systems. Some of the proposed simulation strategies can be found in [4]. For this project, I will be looking for a general solution for any SN P network to be implemented, such that they can then be generated based on a set of initial configuration variables, namely the amount of neurons in a given network, the rules for each network, including the delays per rule, as well as the inclusion of forgetting rules, the synapse connections between neurons and the initial spike configuration for each neuron.

2 Genetic Algorithms with SN P systems

Evolutionary Algorithms have played a large part in computational advancement in recent years. These algorithms were inspired by evolution, as found in nature, for optimization and machine learning[1]. Genetic Algorithms are a section of these Evolutionary Algorithms, initially invented by John Holland in the 1960s, inspired by the adaptation that occurs in nature and aim to replicate that feat in computation. More on the history of genetic algorithms can be found in [1].

In a generic scenario, Genetic Algorithms (GA) have common features prevalent across all implementations, these include the populations of "chromosomes", selection according to fitness, a crossover with the intent to produce new offspring and a random mutation of the generated new offspring.

For the SN P model, the chromosomes in the GA population will take the form of the aspects of the SN P network, where a parent is a fully formed network that provides an output, and a child is an altered network based on two previous parents. These aforementioned aspects are the: amount of neurons in a network, the synapse connections between neurons, the amount of rules within each neuron, the regular expressions which define each rule and the delays on each rule.

The GA will process these populations and replace the subsequent populations, effectively eliminating unfit configurations. This processing will depend upon a fitness function, which will assign a score to each "chromosome". The fitness of a network can be calculated by examining the outputs of each network, which will produce a set of natural numbers. The fitness of each network can be calculated by comparing the target output set T and the generated output set O. There are many considerations on how this fitness could be calculated, the simplest of which being a direct comparison of unique natural numbers in the output set, where a fitness of 1 would be given if sets $T = O$, in the case of $T \subset O$, the fitness would be calculated based on the amount of natural numbers contained in set O that are also in set T. The elements of set O that are not in set T should also be considered, and the ratio of correct elements to incorrect elements should lower the fitness of a given configuration.

In a slightly more comprehensive example, the fitness of network N might be generated by also considering the repetition of elements in set O, as the configuration will be likely to generate certain outputs more often than others. A network which produces elements at a rate as close to T will a higher fitness.

To discuss the crossover between networks, the two selected parent networks that are selected, usually due to their high fitness, which improves their chances at reproduction, would generate a new network configuration based on the five aspects of each network. In general, the fitter the network, the more likely it is to be selected to reproduce in future iterations. Each aspect of the network will need to be considered in the crossover; the amount of neurons, delays and connections are the simplest to alter, however extra consideration will need to be given to altering the regular expressions and the amount of rules within each neuron.

On top of the cross-over between networks, a small chance of mutation should also be added to each generation. This will randomly (non-deterministically) change an aspect of a network configuration, which can appear at any section of the network configuration in the child. The probability of this occurring however should always be very small, and the optimal mutation rate will require further experimentation.

The static variables in a crossover include having only one output neuron, ensuring that there is no neuron left without connections, though a connection to itself is allowed as long as it also connects to another neuron in the network – an open network is not allowed.

The generic structure of the genetic algorithm will take the structure as described in [1]:

"1. Start with a randomly generated population of *n* *l*−bit chromosomes (candidate solutions to a problem).
2. Calculate the fitness *f*(*x*) of each chromosome *x* in the population.
3. Repeat the following steps until *n* offspring have been created:
    a. Select a pair of parent chromosomes from the current population, the probability of selection being an increasing function of fitness. Selection is done "with replacement," meaning that the same chromosome can be selected more than once to become a parent.
    b. With probability *p*c (the "crossover probability" or "crossover rate"), cross over the pair at a randomly chosen point (chosen with uniform probability) to form two offspring. If no crossover takes place, form two offspring that are exact copies of their respective parents. (Note that here the crossover rate is defined to be the probability that two parents will cross over in a single point. There are also "multi−point crossover" versions of the GA in which the crossover rate for a pair of parents is the number of points at which a crossover takes place.)
    c. Mutate the two offspring at each locus with probability *p*m (the mutation probability or mutation rate), and place the resulting chromosomes in the new population. If *n* is odd, one new population member can be discarded at random.
4. Replace the current population with the new population.
5. Go to step 2."

3 Optimization of Control Parameters for Genetic Algorithms

Choosing the optimal parameters for a Genetic Algorithm is an incredibly important task, as it directly influences the performance of the entire model. J. Grefenstette [5] describes the difficulty of controlling a complex process dynamically, especially for system simulation programs, for which the choice of an optimization technique may not be obvious, as well as describing the difficulty of predicting how various parameters interact, such as what effect the increase of population size in response to lowering crossover rate might have.

These problems are especially prevalent in this model, therefore investigating the findings of [5] is especially beneficial when developing a GA model that will fit the SN P system structure. The experimental data in [5] and De Jong's findings in [6] confirm that a mutation rate of above 0.05 are harmful to GA performance, with the performance approaching that of a random search when rates are higher than 0.1. The lack of mutation also seems to point at a loss of performance, suggesting that mutation plays an important role in refreshing lost values in future generations.

Further findings point that a large generation gap and an elitist selection strategy also improve performance. Important considerations for this model are also denoted by the findings regarding population size. For smaller populations, high crossover rates and low mutation rates (or vice-versa) are advised and for mid-sized populations crossover rate appears to decrease as population size increases, which is a fair assumption given that crossover plays an important part in preventing premature convergence [6].

In the case of the SN P system simulation, the optimal population size will need to be found through experimentation. A good example of a high performance, elitist Genetic Algorithm can be found at [7], which describes a Genetic Algorithm that will provide multiple equally optimal solutions. Such an approach could be considered for the SN P simulation, as there could be multiple network configurations that will provide a pareto-optimal output set, rather than a single optimal solution.

One additional thing to consider when implementing Genetic Algorithms that comply with SN P networks is pool selection, as briefly described in section 2. A configuration with the highest fitness would not always be chosen to reproduce, it simply is more likely to do so. Such an implementation would require a large pool size, which would drastically decrease performance in this model. To avoid such performance loss, a better strategy should be chosen. Such strategies could include reject-accept sampling as described in [8], Stochastic universal sampling as described in [9] or Reward-based selection [10] to name a few.

Even though the findings in [5] are very useful for identifying common performance issues, the ideal configurations for the SN P simulation model will require experimentation, as each network will have to take into account multiple additional parameters, such as validity of regular expressions and the actual performance of the SN P simulation.

4 Parallel computing

One of the biggest issues in implanting a simulation of SN P systems is parallelism. The model concerns a high degree of synchronous calculations, which can be very performance intensive to implement on a traditional architecture. When adding Genetic Algorithms to the problem, the computational time may increase by multiple orders of magnitude, as each generation requires the computation of a new SN P system, which will then run for a predefined amount of trials.

To alleviate this performance decrease, multiple algorithms already exist which deal with this problem, such as the Ariadne's Clew algorithm as described in [11]. Designing such a system would be beneficial to the amount of time spent running the simulation, effectively improving the accuracy of the model due to a higher possible number of trials in a given time.

Techniques detailed in [12] can help with the design and implementation of parallel systems, which will compound to give a better performance in the model during execution. The current techniques for synchronising calculations will require additional overheads which will decrease performance, however providing an accurate representation of an SN P model.

An interesting consideration for the development of such a system would be to utilise the highly parallel architecture of a graphical processing unit (GPU) to perform the simulation as described in section 6.6 of [12].

5 Conclusion

Developing a highly parallel simulation of SN P systems is a formidable challenge. To evolve such a system with Genetic Algorithms requires a lot of prior considerations, which many highly accomplished individuals have previously evaluated in their own relevant works. The combination of these ideas can give way to a possible general solution to any output that a potential user might desire an SN P system to provide, which would prove this project to be a success.

The evolutionary nature of Genetic Algorithms in combination with the non-deterministic SN P systems can provide a platform for the study of many interesting branches of computational biology, with a general solution allowing for an automated calculation of any SN P system that might provide a desired output set. An interesting consideration is the repetition of values in a given system. Studying the generations of networks as they converge on a provided output set might prove to point at a certain structure which appears more often, which leads to repeated outputs, which would be an interesting finding that could eventually improve the entire system, if such structures could be carefully avoided in future generations.

The development of this program resembles the development of a weighted graph traversal problem, simply with added parameters and calculation across every traversal. Many findings from previous implementations of such systems can be taken into account when developing this simulation, as they share many common problems.

The visualisation of such a model in action, especially once evolved with Genetic Algorithms and the change between generations has been recorded, would be an excellent way to show the workings of this simulation.

References and citations:
[1] Mitchell M. An introduction to genetic algorithms. MIT press; 1998.

[2] Păun G. Spiking neural P systems. A tutorial. 2007

[3] Luan J, Liu XY. Logic operation in spiking neural P system with chain structure. InFrontier and Future Development of Information Technology in Medicine and Education 2014 (pp. 11-20). Springer, Dordrecht.

[4] Brette, R., Rudolph, M., Carnevale, T. et al. J Comput Neurosci (2007) 23: 349. https://doi.org/10.1007/s10827-007-0038-6

[5] Grefenstette JJ. Optimization of control parameters for genetic algorithms. IEEE Transactions on systems, man, and cybernetics. 1986 Jan;16(1):122-8.

[6] K. A. DeJong, Analysis of the behavior of a class of genetic adaptive systems, Ph.D. thesis, Dept. Computer and Communication Sciences, Univ. of Michigan, 1975.

[7] Deb K, Pratap A, Agarwal S, Meyarivan TA. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE transactions on evolutionary computation. 2002 Apr;6(2):182-97.

[8] Casella G, Robert CP, Wells MT. Generalized accept-reject sampling schemes. InA Festschrift for Herman Rubin 2004 (pp. 342-347). Institute of Mathematical Statistics.

[9] Baker JE. Reducing bias and inefficiency in the selection algorithm. InProceedings of the second international conference on genetic algorithms 1987 Jul (Vol. 206, pp. 14-21).

[10] Loshchilov I, Schoenauer M, Sebag M. Not all parents are equal for MO-CMA-ES. InInternational Conference on Evolutionary Multi-Criterion Optimization 2011 Apr 5 (pp. 31-45). Springer, Berlin, Heidelberg.

[11] Talbi EG, Muntean T. Designing embedded parallel systems with parallel genetic algorithms. InGenetic Algorithms for Control Systems Engineering, IEE Colloquium on 1993 May 28 (pp. 7-1). IET.

[12] L. F. M. Ramos, Developing efficient simulators for cell machines. University of Seville Dpt. of Computer Science and Artificial Intelligence, 2015.