University of Sussex

Top down evolutionary electronics

The "crystallisation" method

Cherry George Mathew

Submitted for the degree of MSc.
University of Sussex
September 2007

Declaration

I hereby declare that this thesis, either in the same or different form, has not been previously submitted to this or any other University for a degree.

Signature:

Cherry George Mathew

UNIVERSITY OF SUSSEX

CHERRY GEORGE MATHEW, MSC.

TOP DOWN EVOLUTIONARY ELECTRONICS
THE "CRYSTALLISATION" METHOD

SUMMARY

This dissertation involves the application of evolutionary techniques to the design of analog electronic filter circuits. A novel new method of tackling complex analog filter design is investigated. It is compared with a standard method used in extrinsic evolutionary electronics for efficiency and scalability. The objective of the project is to demonstrate feasibiliy of the evolutionary method used. It is envisioned that this method can be applied to more complex circuit design problems, and perhaps in domains other than analog frequency filters.

Acknowledgements

Thanks everyone.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Sewall Wright introduced the idea of a fitness landscape to express the relationship between genotype, phenotype and viability, in what would later be known as the field of theoretical population genetics. [wright,1932]. Borrowing from these ideas, the field of Evolutionary electronics has attempted to pose the problem of circuit design as one of search along a suitable fitness landscape. If the topology and component specifications of a circuit can be arranged to be the phenotypic expression of a string of coding genes, then an evolutionary algorithm need only search for circuits which behave in increasingly conformant fashion, in order to meet a required design goal.

There are two methods to search for conformance to a desired specification. The first involves actual realisation of each physical circuit phenotype in hardware, followed by testing its behaviour for conformance. Modern programmable hardware such as FPGAs and FPTAs make this method feasible and extremely fast. The main caveat, or advantage of this method, depending on one's point of view, is that the evolutionary method tends to use the physics of the given hardware and the environment at the time of evolution in unforeseen ways, bringing novel, but usually unreproducible designs (Thompson, 1997). The second method involves the use of simulators to realise circuit behaviour. With the obvious disadvantage of lack of speed and emulator approximation, comes the advantage of being able to reproduce results eg: (Koza et al., 1997). This project uses the circuit emulator SPICE, in order to evaluate candidate circuits.

From a circuit design perspective, the main challenge facing evolutionary electronic design has been the design of increasingly complex circuits. Various methods of constraining evolutionary pathways have been proposed to allow for realistic design times, given limited computational resources. eg: [kozagp3], (Kalganova, 2000; Torresen, 2002)

This project approaches the problem in a top-down fashion. By starting with a circuit mesh connected in all possible ways and circuit elements which have an amalgam of properties of individual elements, it is expected that the evolutionary fitness landscape is smoothened. The topology is discovered by disconnecting initially connected nodes as required (ie; introducing parsimony), and circuit elements of required specification are arrived at by progressively weakening the contribution of the properties of all but the desired circuit element. This approach can be thought of as analogous to crystallisation of minerals from a saturated solution, where the electrostatic bonds between the mineral molecules are constrained to the physical bounds of the geometry.

As an illustrative project, only circuits with passive two port elements are used for crystallisation. Arbitrary target frequency filter patterns are specified as evolutionary goals to the setup. Results are analysed and compared to a naive approach of having all components inter-connected at the start of the run, but with standard two port circuit elements in place.

Chapter 2

Background and Motivation

Evolutionary Electronics has been investigated by many people over the years. John Koza has demonstrated industrial strength circuit designs and even innovative new designs by Genetic Programming methods (Koza et al., 2004) and predicts the success of these techniques as computing power increases. Zebulum surveys various approaches to evolutionary electronics (Zebulum et al., 1998) and discusses issues specific to the problem of analog filter design. Mentioned in this survey are work by Grimbleby [grimbleby95] which involves the discovery of the circuit topology by an Evolutionary Algorithm, and the circuit component values, by numeric optimisation; Horrocks and Spittle[horrocksspittle96]: and Horrocks and Khalifa[horrockskhalifa96]: whose work involves active filters and choosing ”preferred values ”, taking into account parasitic effects. Zebulum addresses a third issue, namely the need to seek for parsimonious solutions in the face of increasing circuit complexity. The analogous approach in the Genetic Programming approach is pointed out to be to ”control the depth size of the trees”, referring to the acyclic graphs representing GP -¿ circuit mappings. In the GA approach of Zebulum, circuit complexity is increased by allowing for gene strings to increase in length, and using the SAGA approach (Harvey, 2001) to search along ”neutral ridges” for higher complexity.

In this work we focus on approaching the problem the other way round. Instead of trying to increase complexity, we start with a complexity with an upper limit, and work ourselves downwards to less complex solutions, whilst enforcing parsimony in phenotype in a special manner dubbed ”crystallisation” and referred to as such henceforth in this document.

Chapter 3

Methodology

3.1 Approach

The two main aspects of any circuit design are the topology of the network of interconnected devices, and the values of the component devices themselves.

3.1.1 Topology

Given a completely interconnectable mesh of components, potentially all possible topologies are available as a superset of the required circuit topology (or set of viable circuit topologies). We focus on the problem of passive frequency filter design. Consequently, we consider only two port devices, specifically resistors, capacitors and inductors. There is no reason why this cannot be extendable to active three port devices applied to the same problem, or even other circuit design problems. The current design problem is only picked as a relatively simpler illustrative example of the approach.

Having started with a configuration in which devices are connected in every possible permutation, the evolutionary algorithm searches for two parameters: functional correlation and parsimony.

Function Correlation

A target function is defined, which has a pre-defined frequency response. The frequency response of every candidate circuit is assessed by running it through the SPICE [berkeleyspice] simulator and correlating the sampled frequencyresponse returned by the simulator with the target function. The higher the correlation, the better the circuit behaves according to our specification, and therefore is deemed fitter.

Parsimony

Eventually, we intend that the final circuit uses the minimum possible number of components in the most efficient manner as possible. In order to encourage this, we award higher fitness to circuit topologies with fewer elements.

3.1.2 Circuit Elements

Individual elements are picked from a set of ”preferred values”. The actual values in the current work are arbitrary and only meant to reflect a wide range of values. They do not represent prac-

tically realisable values or fabrication limitations. A virtual circuit element dubbed a ”unitor” is defined. This is the key to the ”crystallisation” method. The unitor is an amalgam of a resistor in series with a capacitor and an inductor in parallel **??**. One of the key objectives of the evolutionary algorithm is to ”crystallise” the unitor into a single real circuit element. Table **??** shows the configurations in which this may be accomplished. As an illustration, consider a unitor which is aiming to be a resistor. The easiest way to do this is to short circuit the inductor OR the capacitor elements within the unitor. This condition is recognised within the code and specially handled so that the unitor is now defined exclusively as a Resistor. A fitness method is applied to every individual unitor encouraging it to transit to a single device element. This is done by returning highest fitness for unitors which consist of single realisable devices, intermediate fitness for two element unitors, and lowest fitness for unitors which still have all three elements in them. This fitness aspect is accumulated over all the unitors of the entire circuit and contributes to the increase parsimony (decrease in total device count) in the final circuit.

Chapter 4

Implementation

The implementation can be described best by breaking it down into areas of functional implemenation namely: the Genetic Algorithm, Gene encoding, the netlist generator, the spice simulator driver, and the fitness function; each of which are explained in some detail below.

4.1 The Genetic Algorithm - Microbial GA

The Genetic Algorithm used here is the microbial GA (Harvey, 1996). Although a detailed analysis of this algorithm is out of the scope of this dissertation, a brief description of this algorithm is in order.

The microbial GA may be regarded as a steady state, rank-based tournament selection algorithm with "horizontal" gene exchange. The key to selection is that in a tournament, elitism is preserved by ensuring that tournaments always leave winners with highest fitness in the current population intact. Cross-over and mutation occur during each tournament with fixed probabilites, and the loser is eliminated from the population thus providing automatic selection bias towards fitter individuals. A "generation" in the conventional sense consists of a set of tournaments equal to the population size of the GA.

4.2 Gene encoding

The gene encoding used in the implementation is very simple. A genestring of fixed length is defined. Genes coding for a given phenotype (a node, a resistor, capacitor or inductor) can be of arbitrary size as required to represent all phenotypic ranges. The nucleotide coding equivalent is binary, represented by single bits, as in standard Genetic algorithms (Goldberg, 1989).

The position of the gene along the genestring defines the phenotype it represents. In our implementation, the first gene represents the circuit node chosen as the output node. The length of this gene depends on the number of bits required to represent the highest node number. Subsequent genes represent unitors in increasing sorted order of the permutation of their connecting nodes. Eg: A unitor connected to node 2 and 3, U23, is coded on the genestring before U34. U21 comes before U23.

XXX: figure with gene representation.

Each Unitor is encoded as a bit array of six nibbles [1]. Each pair of nibbles represents a component specification followed by its importance, termed "device bias" in the unitor. Three such

[1] a nibble is 4 bits

pairs represent resistance, capacitance and inductance respectively.

XXX: figure for gene representation of each unitor.

The value of each component specifier nibble is an index into a lookup table from where the netlist generator code fetches the actual value of the specification of the device to be used. The nibble representing device bias has no direct effect on the phenotype (ie; the final circuit netlist), but is very important for the crystallisation method (see below.)

Finally, every gene is decoded through a Gray to binary decoder to ensure smoother hamming distances between mutations (Goldberg, 1989). However, the extent to which this has affected this particular problem has not been investigated.

4.3 The netlist generator

This is the core of the simulator, and can be viewed as the "morphogenerator" for the genes. The netlist generator parses a given gene string, and writes out a file containing a SPICE [berkeleyspice] netlist corresponding to it. This netlist is then run through the ngspice [ngspice] simulator and analysed. The ngspice simulator runs a frequency domain analysis of the given network, obtaining the transfer function as a relationship between the input frequency and the output Voltage (complex values). The netlist generator arranges for this information to be extracted at discrete sample points over the frequency domain by means of the SPICE .PRINT command.

For every individual netlist produced, this procedure is carried out, followed by a fitness evaluation. Every netlist generation involves an iterative loop of all possible combinations of node pairs to which unitors are attached. Each unitor is generated by reading off and decoding its corresponding genome, and generating a .SUBCKT SPICE routine with the unitor components inside. The code which does this is intelligent enough to detect when the unitor should be a short or open circuit. As required, it will either short circuit, or entirely skip the synthesis of the unitor for a given node pair. The parsimony fitness aspect is zero for a short circuit, and maximum for an open circuit, with intermediate values assigned, as described below.

Finally, every unique individual is timestamped and written out to disk before simulation in SPICE. This allows for the SPICE simulator driver to work independantly (but not parallelly) of the GA setup, while at the same time preserving the individual circuit for subsequent inspection. Although wastefull of disk space, this is easier than saving individual gene strings and subsequently regenerating the netlist as required.

4.4 The SPICE simulator driver

Although the details of its implementation are unimportant, it is worth mentioning that the simulator driver uses the unix fork(3) and exec(3) calls to invoke the ngspice simulator with the appropriate netlist. The simulator results are then accumulated, fed into suitable data structures, and fed back to the calling routine to pass on to the fitness evaluation routine.

4.5 The fitness function

The fitness function is the crucial element of any genetic algorithm. Given that the fitness function reduces multiple fitness criteria into a single, scalar fitness landscape, it is crucial to design the fitness function carefully. Ideally, a carefully designed fitness function would have gentle (non-

rugged) landscapes with provision for neutral ridges and few local minima. This would enable the GA to converge smoothly to the ideal solution we seek.

There are three broad fitness criteria in unitor based circuits: The transfer function target correlation, the proximity of the component unitors to a single device, and general parsimony of component devices.

The fitness function combines these three aspects into a single scalar value which the evolutionary algorithm then seeks to maximise. The base correlation fitness is first calculated and then altered by the parsimony fitness. Variations of this theme have been tried with limited results.

4.5.1 Transfer function correlation

The objective of the GA is to find a circuit which can produce a frequency response as close to a specified target frequency response as physically possible, by choosing from the available component mix. The correlation is acheived based on previous work. (Zebulum et al., 1998)

XXX: Equation: fitness = SUMOF(weight[i] * abs(target[i] - output[i]))

An array of weights and targets is initialised beforehand based on estimation of the required target response. The weights are assigned such that they are much lower value in the signal reject frequency ranges than they are in the signal pass frequency range. The exact ratio depends on the number of target sample points and the nature of the target signal. We have used the same weights for lowpass and highpass filters for all frequencies, while for bandpass and bandreject targets we use weights 1/32 times that in the bandpass region. As the Q-factor of bandpass/reject filters increases, it is best to increase the weight ratio, however if the ratio is too high, the band reject areas tend to fall out of line, since their deviation from the target is then not checked with sufficient penalty.

4.5.2 Unitor properties

The fitness awarded to each individual unitor fits on a 0 - 100 scale. Associated with every unitor is a set of three values indicating its proximity to becoming an individual device, already dubbed "device bias". As mentioned above, these values are completely free running and bear no correlation to the actual specification of the component devices that make up a unitor. They however directly affect the fitness of the unitor. Two special cases occur when these values actually affect the specification of the component devices. At 0 and 15 respectively (corresponding to the lowest and highest values respectively encodable in a nibble), the values indicate to the netlist generator that the device is either a short-circuit, or to be open-circuited (ie; not generated). This correlates directly with the parsimony fitness requirement (see below) that fewer components are encouraged over lots of short circuited nodes.

Every unitor also has three separate gene encoded values (see 4.2) to indicate the physical values that each component device takes up. This is realised in the actual netlist and directly affects the transfer function of the entire circuit, as explained above.

4.5.3 Parsimony

The overall fitness of the circuit is arranged to improve as the number of open circuited nodes increases. However, this needs to be balanced carefully with the correlation fitness, as unbounded opening of circuits leads to non-functional circuits.

4.6 Benchmarking

The results obtained with this method are meaningless without a suitable comparison with a standard technique. A simple alternative circuit simulation is setup for this purpose. The code is arranged such that all but the most essential code is shared from the same source files. This assures a level of confidence in the results. The alternative arrangement involves the simplification of the unitor netlist generator. Instead of three devices and associated device bias, we encode a single gene of 6 bits for a single device. The lookup tables from which the devices are chosen are a union of those used for the unitor. The algorithm can recognise the type of device (resistor vs. capacitor vs. inductor) based on the index depth. Since 6 bits can represent upto 2^6 indices and the LUT has fewer entries than this, the extra information is uniformly distributed in the index range by scaling the decoded gene value with the maximum required index. This ensures that rollover does not favour a certain type of device as would be the case with division or modulo operation.

The arrangement in this simulator setup is referred to henceforth as the ”naive” method.

Chapter 5

Results

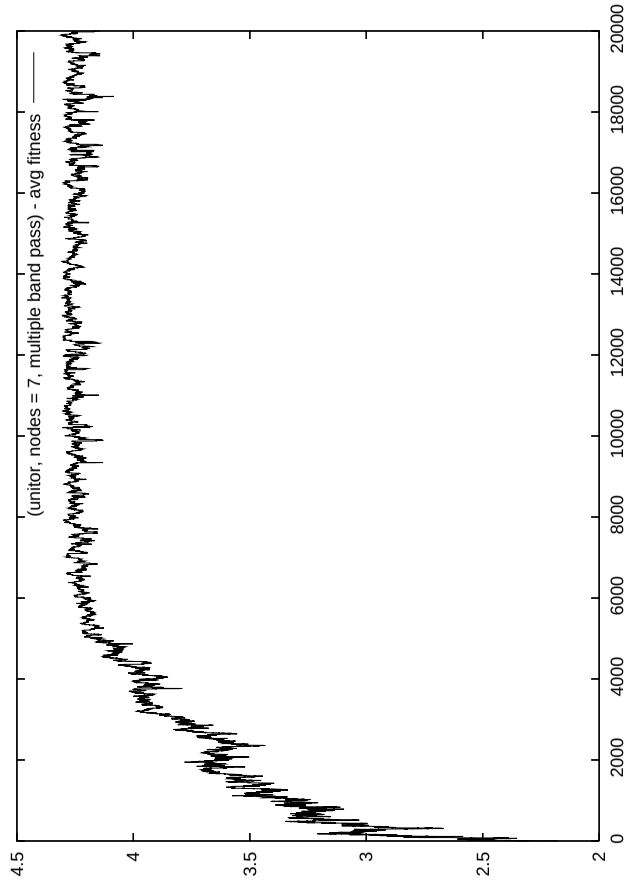

Figure 5.1: Average Fitness of the population: Unitor, Nodes = 7

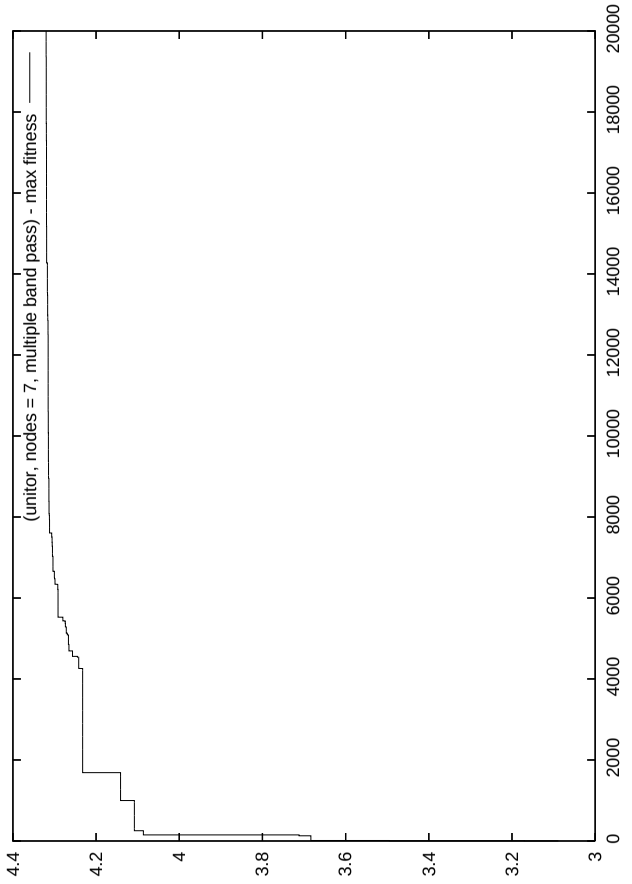

Figure 5.2: Fitness of individual with maximum fitness in the population: Unitor, Nodes = 7

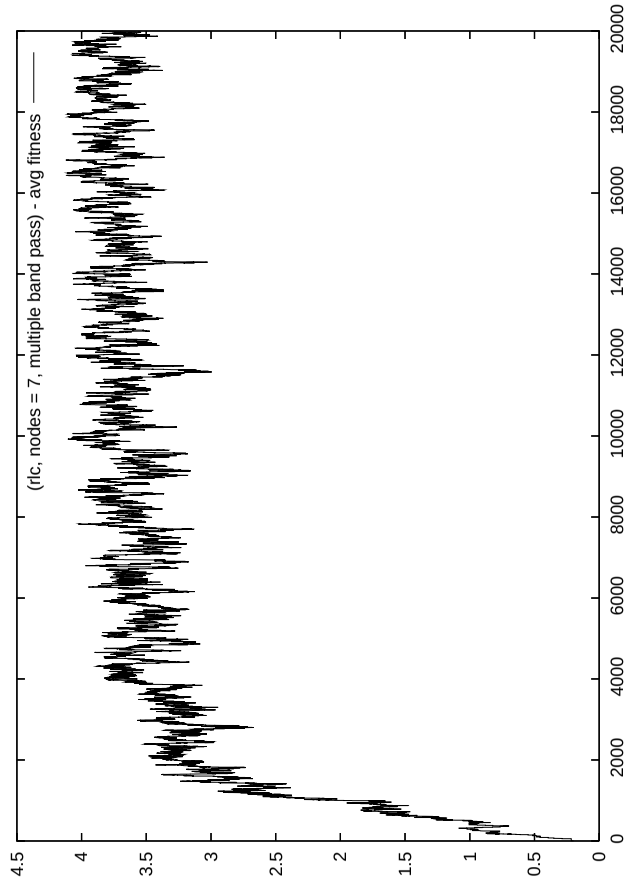

Figure 5.3: Average Fitness of the population: RLC, Nodes = 7

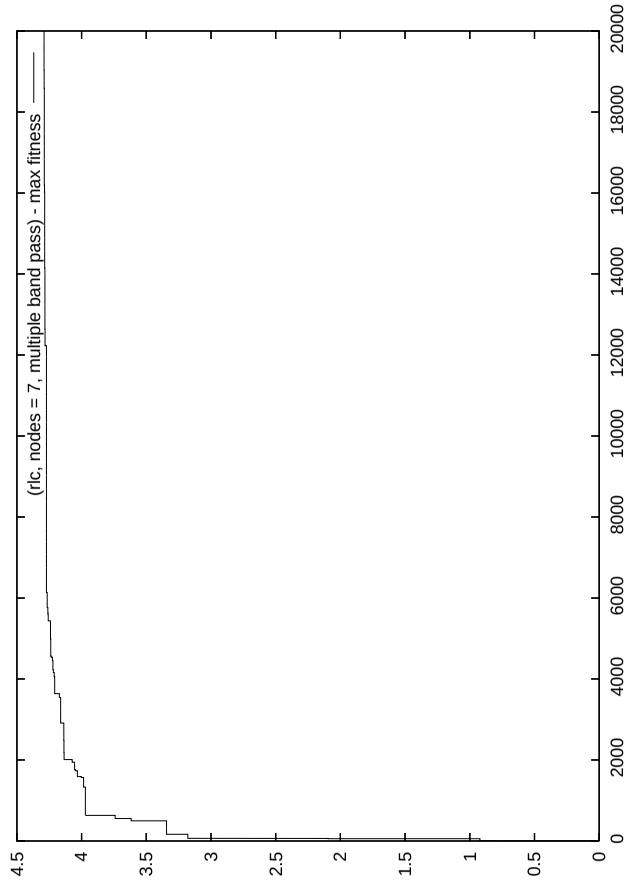

Figure 5.4: Fitness of individual with maximum fitness in the population: RLC, Nodes = 7

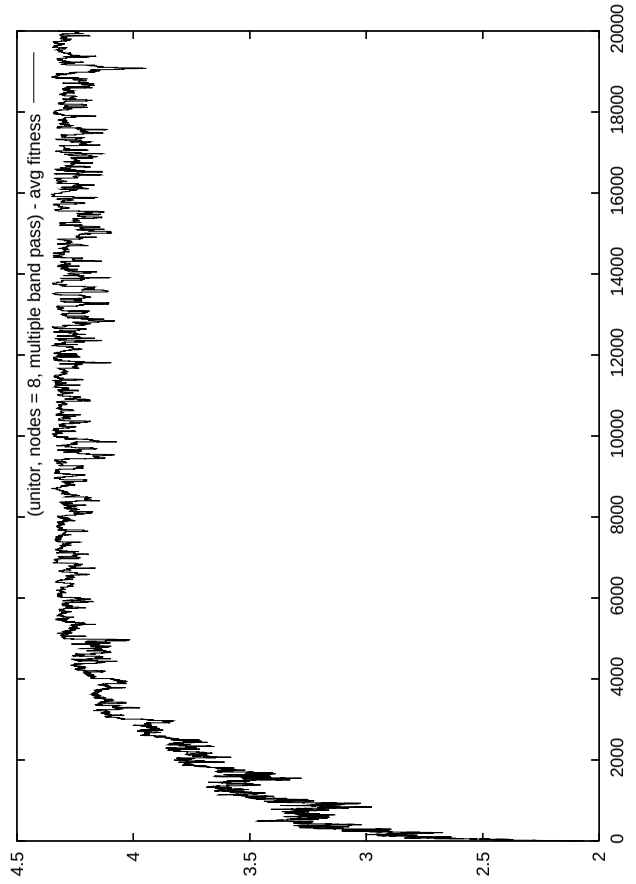

Figure 5.5: Average Fitness of the population: Unitor, Nodes = 8

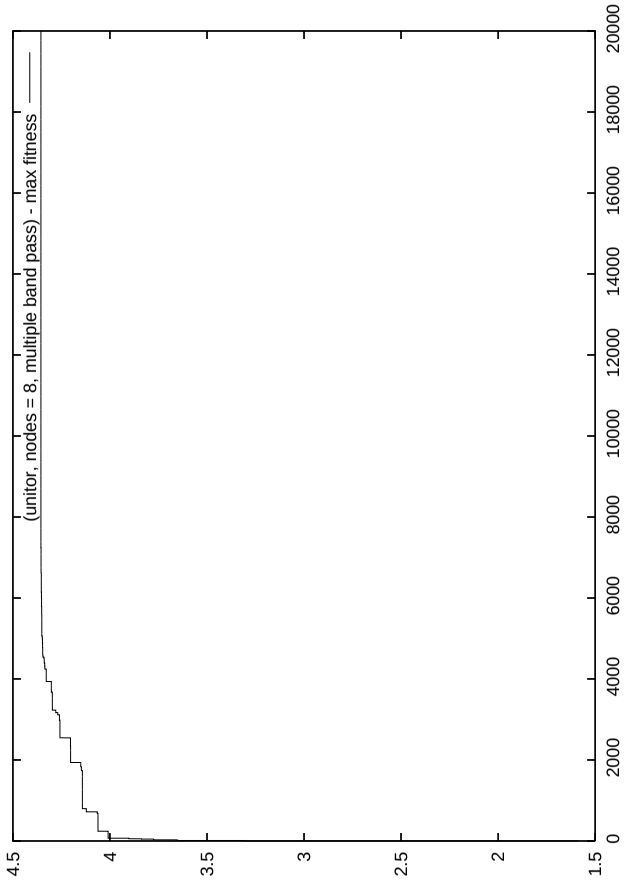

Figure 5.6: Fitness of individual with maximum fitness in the population: Unitor, Nodes = 8

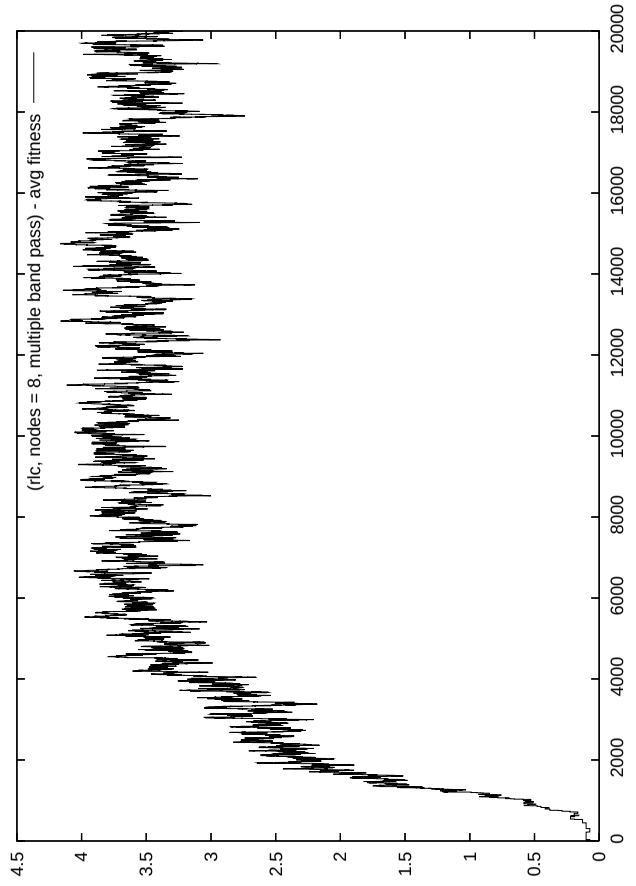

Figure 5.7: Average Fitness of the population: RLC, Nodes = 8

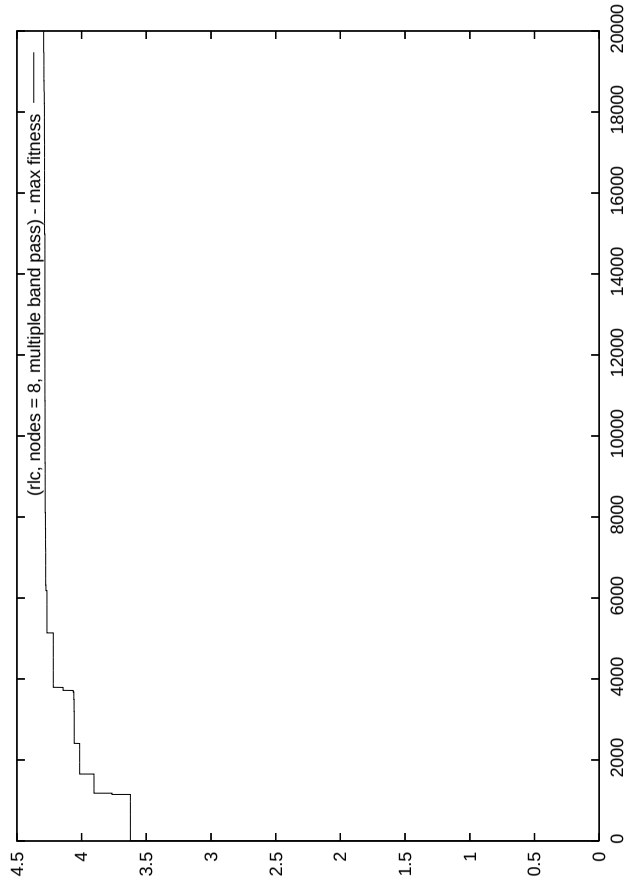

Figure 5.8: Fitness of individual with maximum fitness in the population: RLC, Nodes = 8

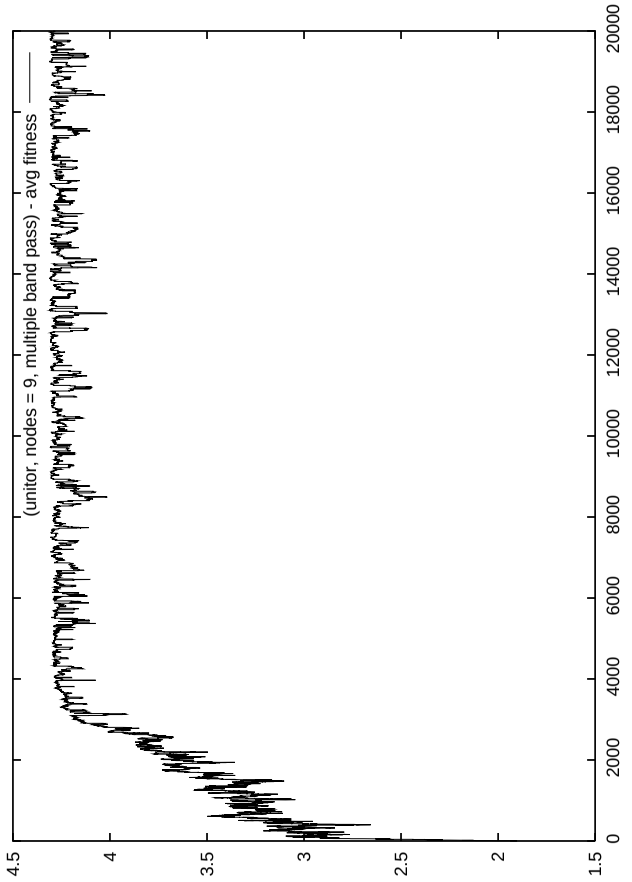

Figure 5.9: Average Fitness of the population: Unitor, Nodes = 9

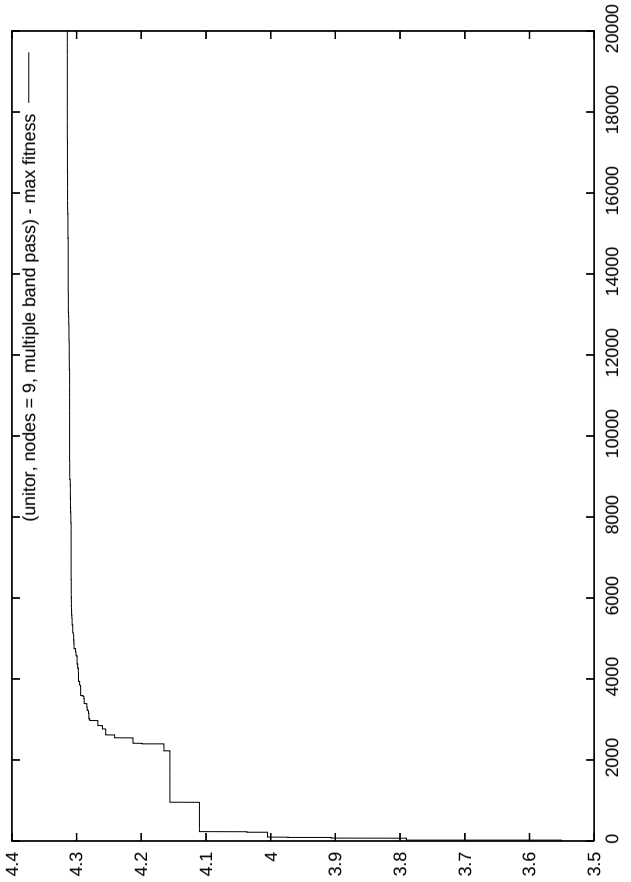

Figure 5.10: Fitness of individual with maximum fitness in the population: Unitor, Nodes = 9

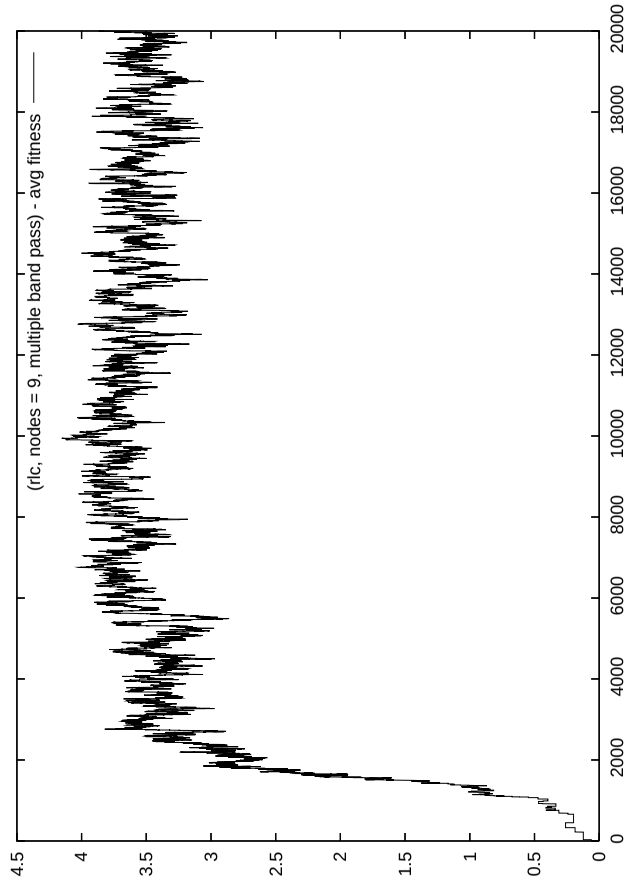

Figure 5.11: Average Fitness of the population: RLC, Nodes = 9

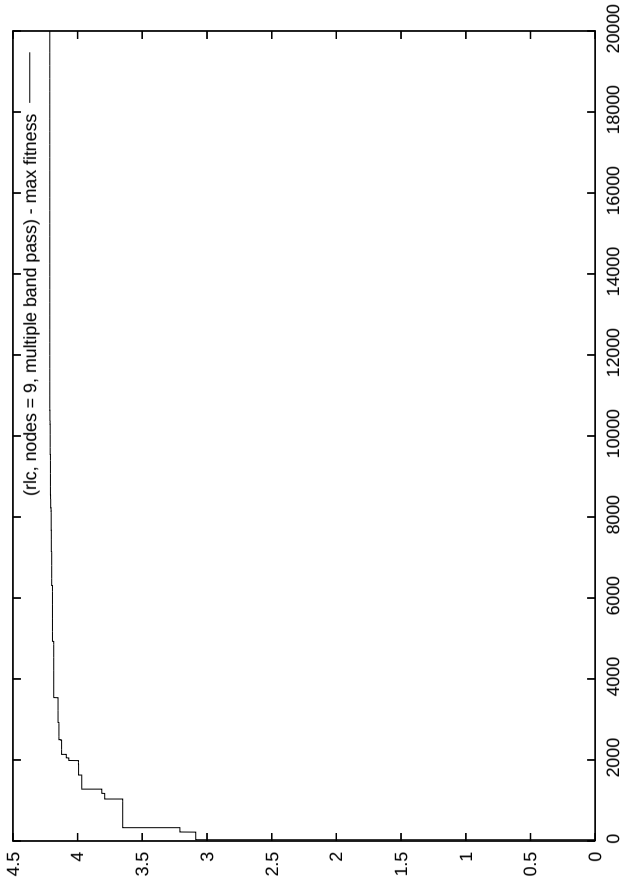

Figure 5.12: Fitness of individual with maximum fitness in the population: RLC, Nodes = 9

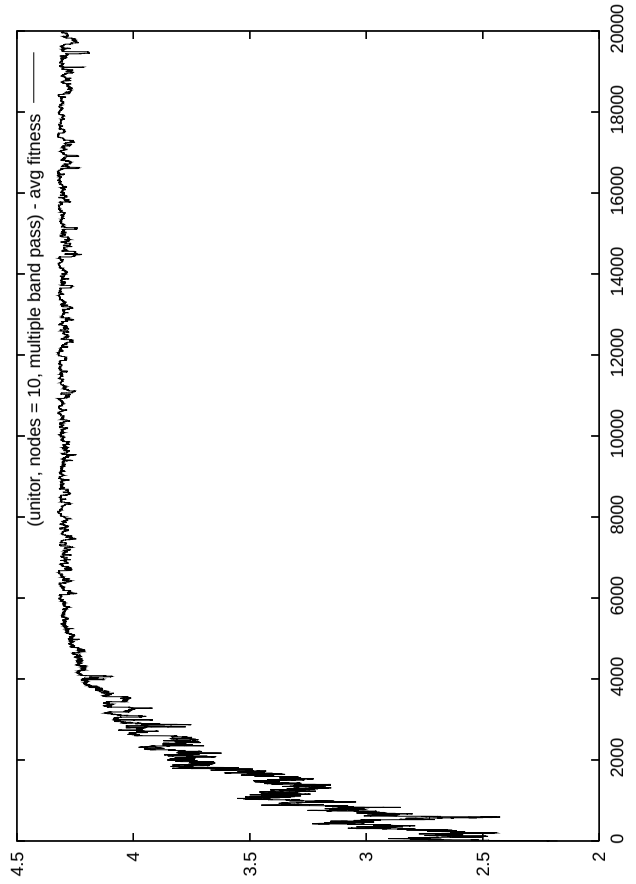

Figure 5.13: Average Fitness of the population: Unitor, Nodes = 10

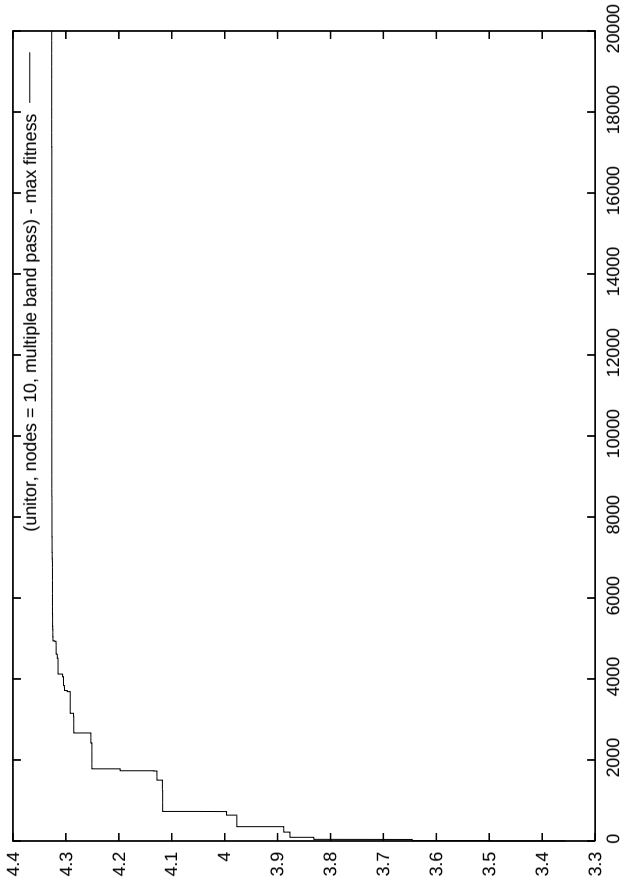

Figure 5.14: Fitness of individual with maximum fitness in the population: Unitor, Nodes = 10

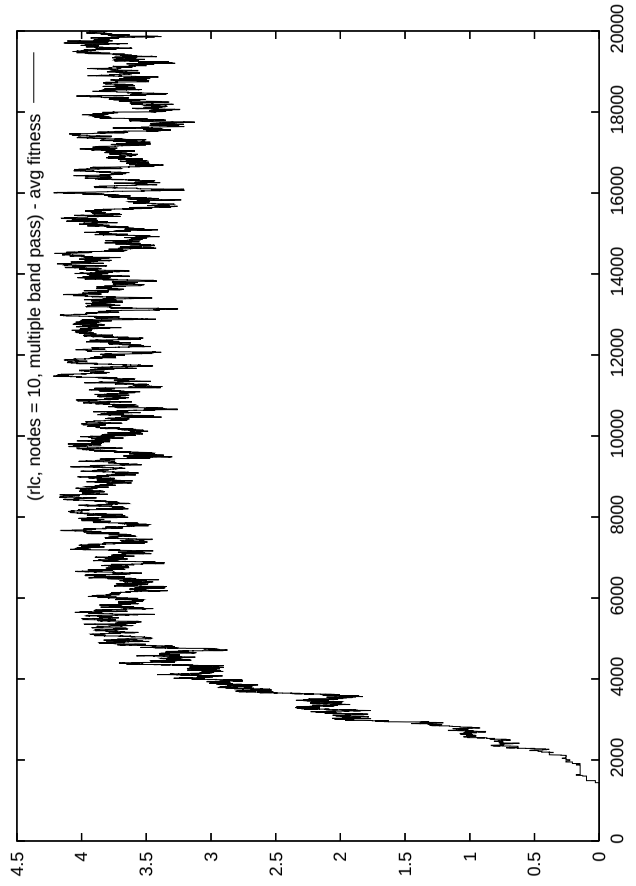

Figure 5.15: Average Fitness of the population: RLC, Nodes = 10

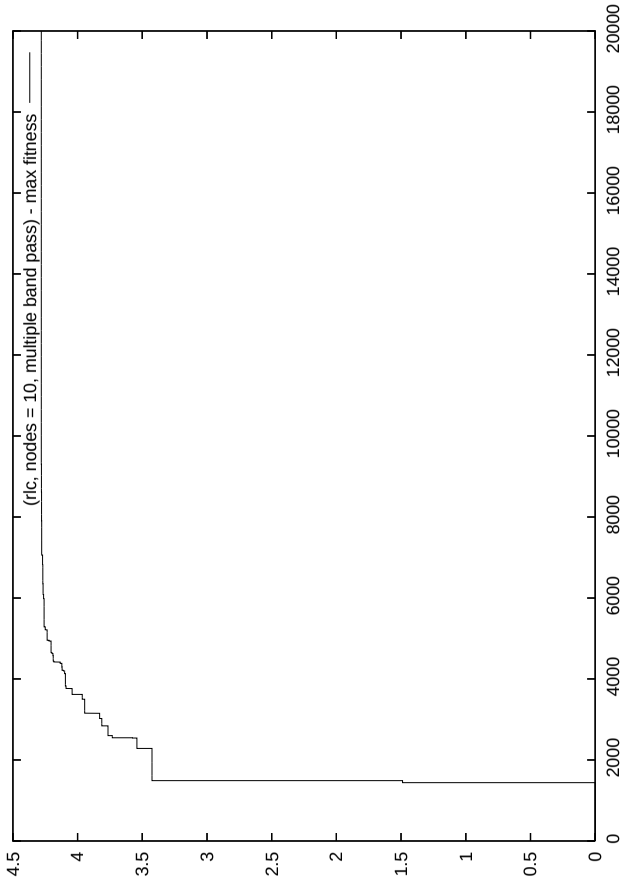

Figure 5.16: Fitness of individual with maximum fitness in the population: RLC, Nodes = 10

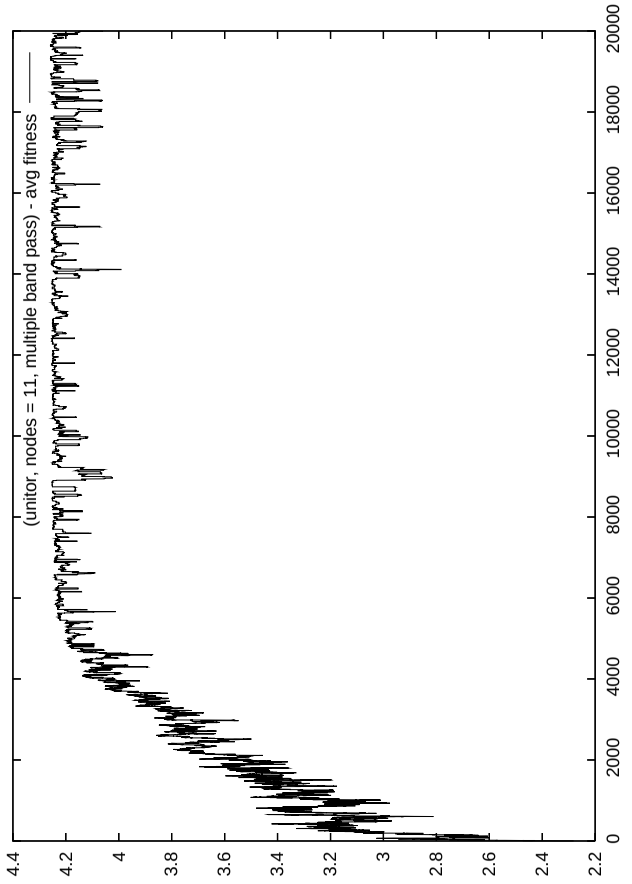

Figure 5.17: Average Fitness of the population: Unitor, Nodes = 11

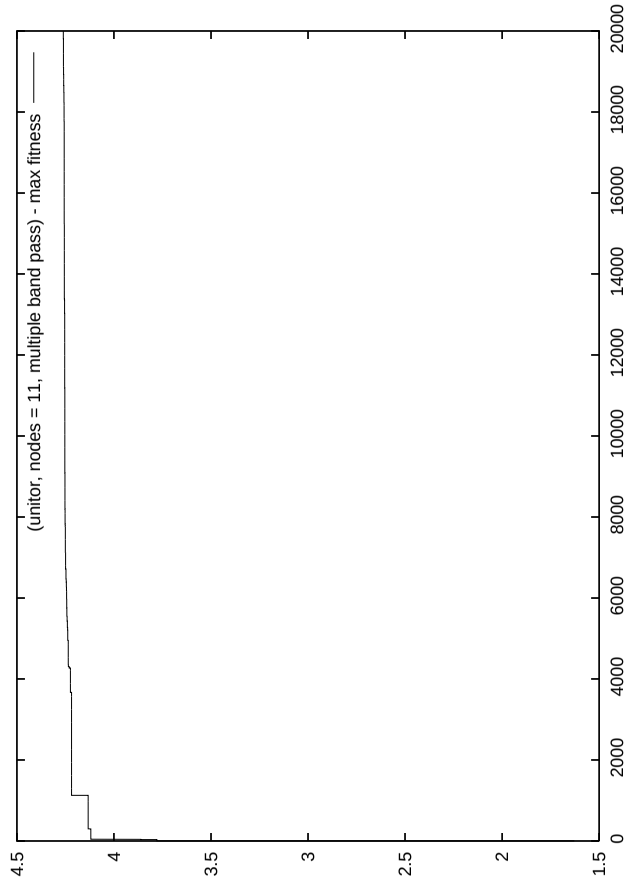

Figure 5.18: Fitness of individual with maximum fitness in the population: Unitor, Nodes = 11

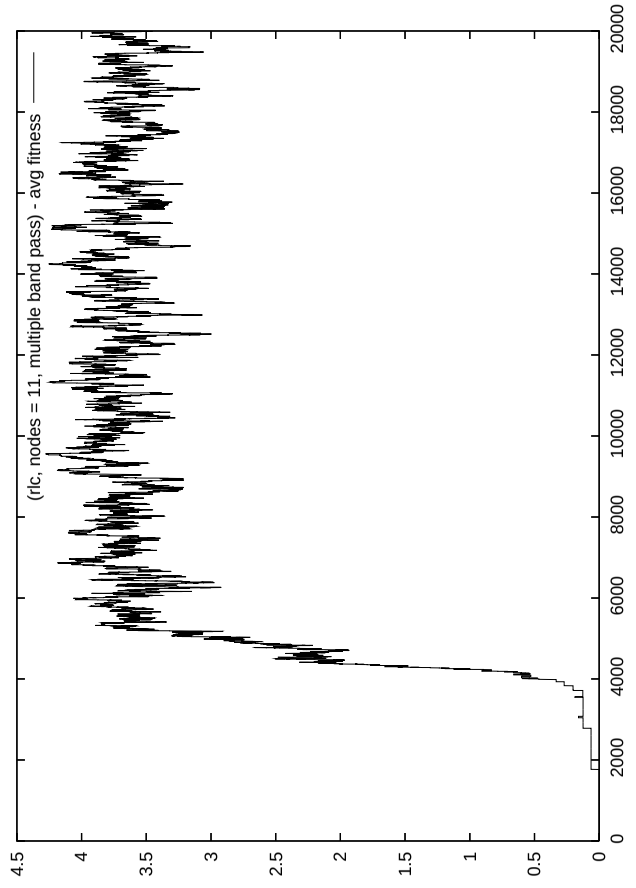

Figure 5.19: Average Fitness of the population: RLC, Nodes = 11

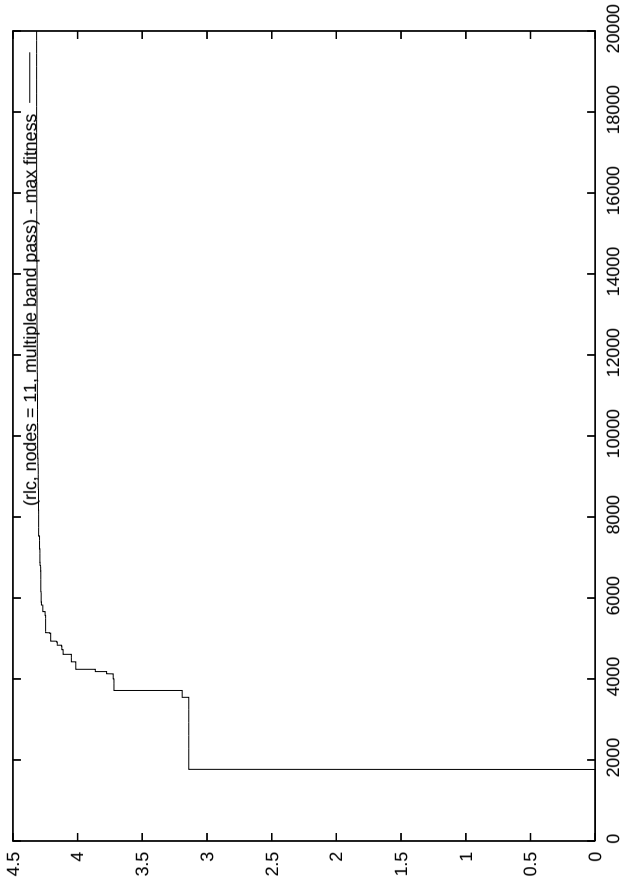

Figure 5.20: Fitness of individual with maximum fitness in the population: RLC, Nodes = 11

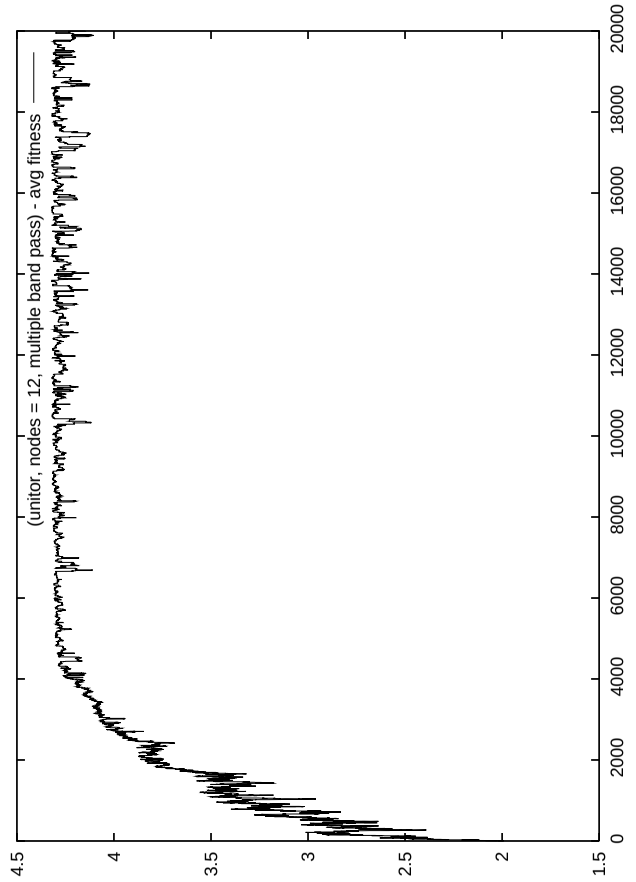

Figure 5.21: Average Fitness of the population: Unitor, Nodes = 12

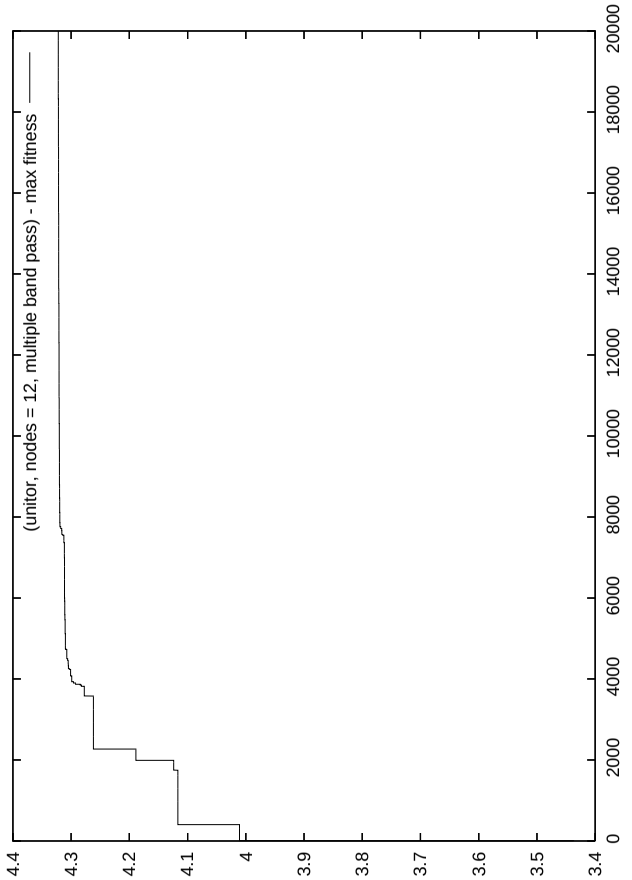

Figure 5.22: Fitness of individual with maximum fitness in the population: Unitor, Nodes = 12

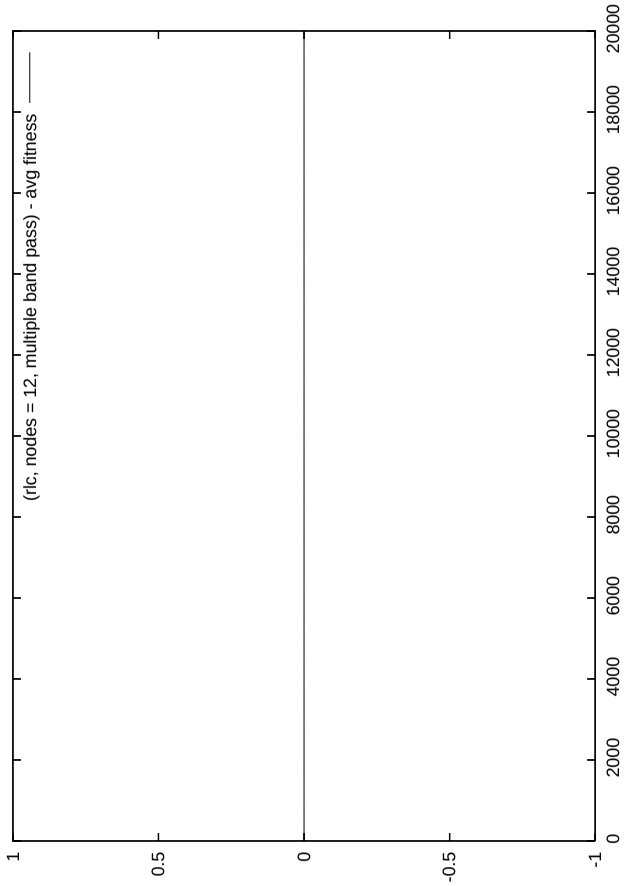

Figure 5.23: Average Fitness of the population: RLC, Nodes = 12

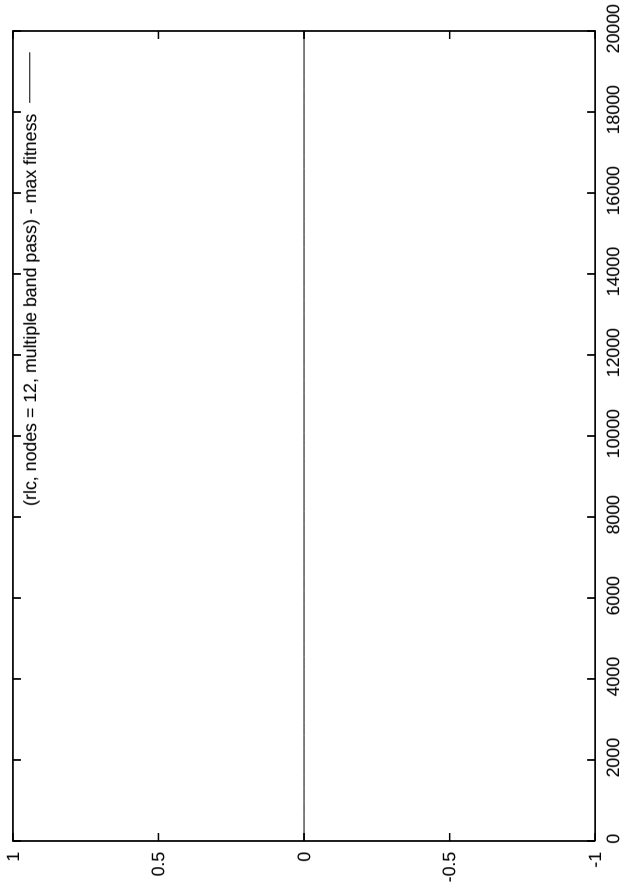

Figure 5.24: Fitness of individual with maximum fitness in the population: RLC, Nodes = 12

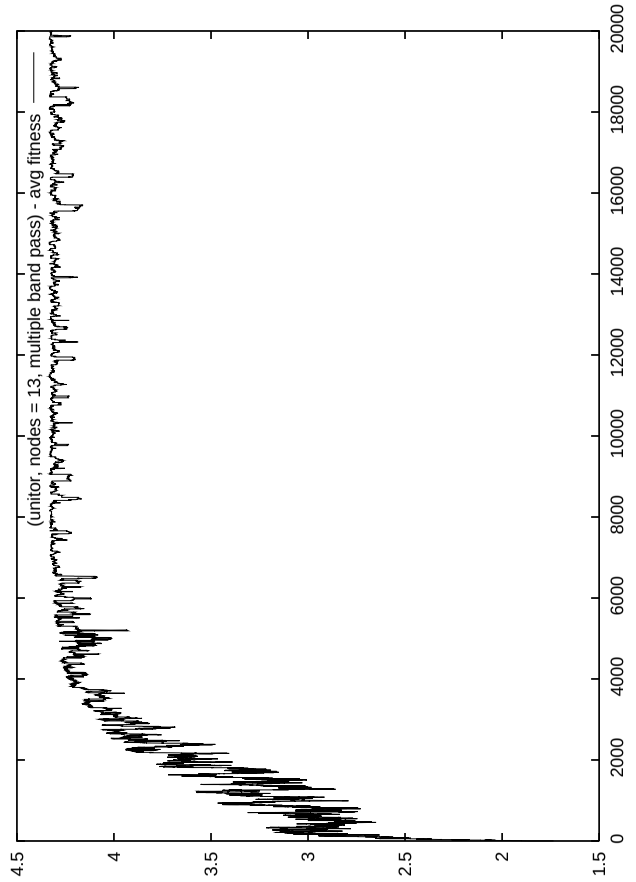

Figure 5.25: Average Fitness of the population: Unitor, Nodes = 13

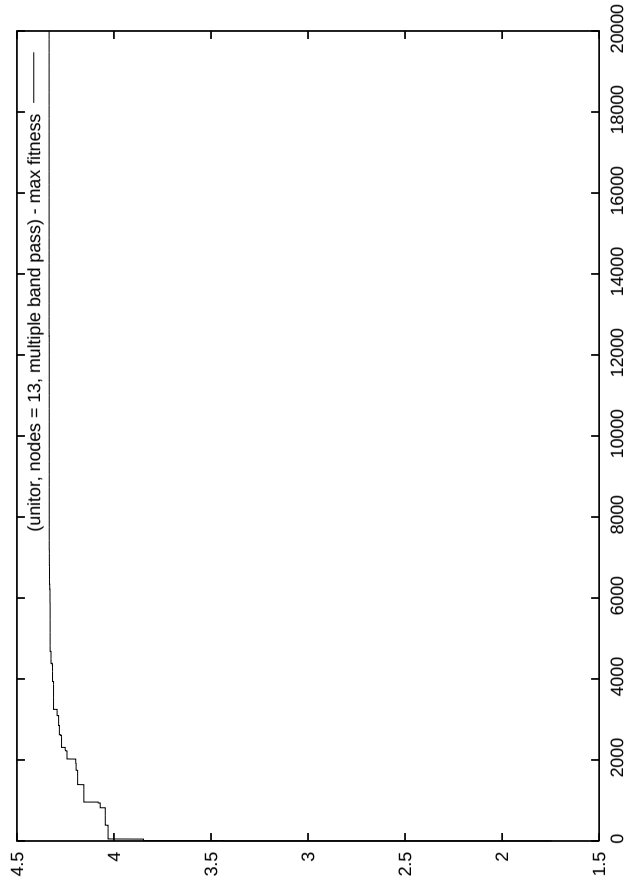

Figure 5.26: Fitness of individual with maximum fitness in the population: Unitor, Nodes = 13

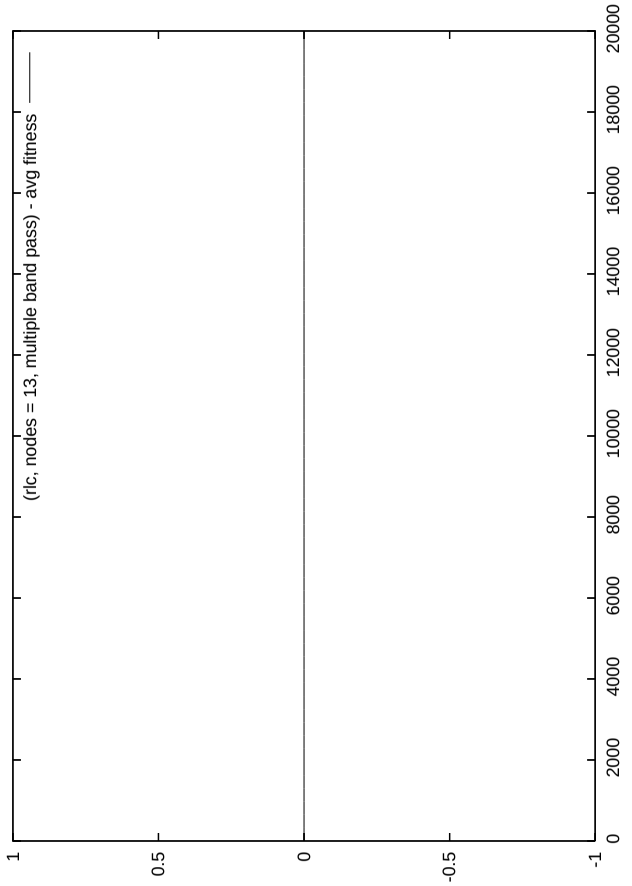

Figure 5.27: Average Fitness of the population: RLC, Nodes = 13

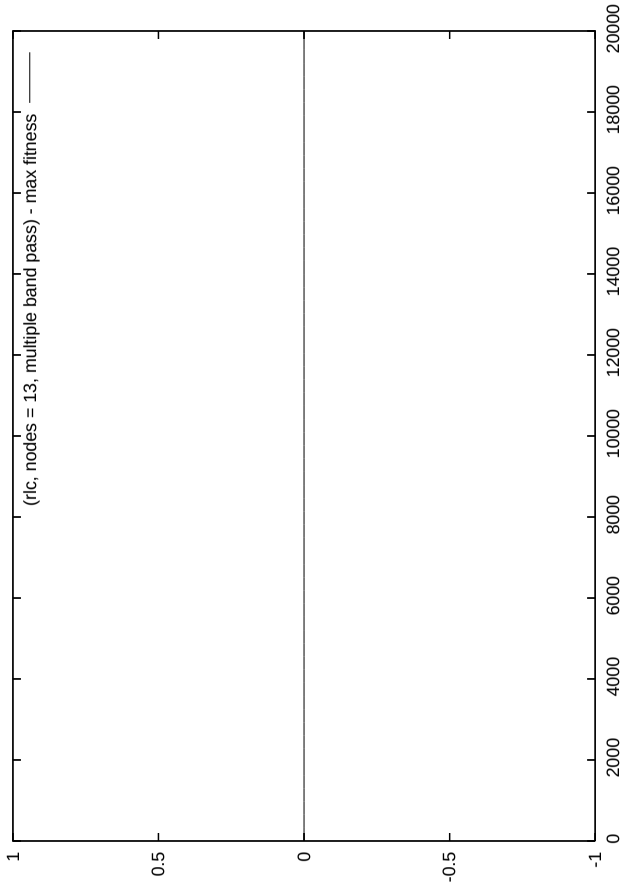

Figure 5.28: Fitness of individual with maximum fitness in the population: RLC, Nodes = 13

Chapter 6

Discussion and further work

The most interesting observation for our discussion borders around the fact that the fitness landscape due to the crystallisation method is much smoother than that of the ”naive” method. The fitness landscape picks up almost from the start, with no sharp fitness ”jumps”. This is in contrast to the fitness plot of the ”naive” method, which takes a while to discover the right combination of devices to kick-start the hill-climb. Since the device arrangement of the ”naive” method is very similar to that of traditional intrinsic Evolutionary Hardware efforts on transistor arrays (Thompson, 1996), [layzell:2001], it would be tempting to argue that the ”search” method for the right transistor configuration could be enhanced by incorporating the ”crystallisation” technique. For example, Fig. 6.6 in [layzell:2001] shows almost zero fitness for a major portion of the amplifier runs before the right configuration to bias the transistor in the active is discovered. Since transistor operation in the active region is a prerequisite for its operation as an amplifier, there seems to be a direct correlation with the jump in fitness mentioned, and the discovery of the right topology and circuit components required for this to happen. This provides a strong argument that the crystallisation technique can be used to speed up discovering these configurations with active devices.

Using the crystallisation method to transistor arrays is obviously impractical as an exclusively intrinsic method since the circuits begin in a completely interconnected fashion at the point of commencement of evolution. This would invariably result in dangerous configurations such as forward biased gate-¿drain configurations which could destroy the device array. Consequently, an extrinsic method, or a combination method would be the best way to investigate the crystallisation method with active devices.

Chapter 7

Conclusion

XXX: Conclude the chapter here.

Bibliography

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. 5, 6

Harvey, I. (1996). The microbial genetic algorithm. 5

Harvey, I. (2001). Artificial evolution: A continuing SAGA. *Lecture Notes in Computer Science*, 2217:94–?? 2

Kalganova, T. (2000). Bidirectional incremental evolution in extrinsic evolvable hardware. In Lohn, J., Stoica, A., and Keymeulen, D., editors, *Proceedings of the Second NASA/DoD Workshop on Evolvable Hardware (EH2000)*, pages 65–74, Palo Alto, California, USA. 1

Koza, J. R., Bennett III, F. H., Andre, D., Keane, M. A., and Dunlap, F. (1997). Automated synthesis of analog electrical circuits by means of genetic programming. *IEEE Transactions on Evolutionary Computation*, 1(2):109–128. 1

Koza, J. R., Jones, L. W., Keane, M. A., and Streeter, M. J. (2004). Towards industrial strength automated design of analog electrical circuits by means of genetic programming. In O'Reilly, U.-M., Yu, T., Riolo, R. L., and Worzel, B., editors, *Genetic Programming Theory and Practice II*, chapter 8, pages 120–?? Springer, Ann Arbor. pages missing? 2

Thompson, A. (1996). Silicon evolution. In Koza, J. R., Goldberg, D. E., Fogel, D. B., and Riolo, R. L., editors, *Genetic Programming 1996: Proc. 1st Annual Conf. (GP96)*, pages 444–452. Cambridge, MA: MIT Press. 38

Thompson, A. (1997). An evolved circuit, intrinsic in silicon, entwined with physics. In Higuchi, T., Iwata, M., and Weixin, L., editors, *Proc. 1st Int. Conf. on Evolvable Systems (ICES'96)*, volume 1259 of *LNCS*, pages 390–405. Springer-Verlag. 1

Torresen, J. (2002). A scalable approach to evolvable hardware. *Genetic Programming and Evolvable Machines*, 3(3):259–282. 1

Zebulum, R. S., Pacheco, M. A., and Vellasco, M. (1998). Comparison of different evolutionary methodologies applied to electronic filter design. In *Proceedings of the 1998 IEEE World Congress on Computational Intelligence*, pages 434–439, Anchorage, Alaska, USA. IEEE Press. 2, 7

Appendix A

Code

```
10 PRINT "HELLO WORLD"
```