Chapter 1

Issues in the Scalability of Gate-level Morphogenetic Evolvable Hardware

Justin Lee and Joaquin Sitte

Smart Devices Laboratory
Faculty of Information Technology
Queensland University of Technology
GPO Box 2434, Brisbane, Qld 4001, Australia
E-mail: jm.lee@qut.edu.au, j.sitte@qut.edu.au

Traditional approaches to evolvable hardware (EHW), in which the field programmable gate array (FPGA) configuration is directly encoded, have not scaled well with increasing circuit and FPGA complexity. To overcome this there have been moves towards encoding a growth process, known as morphogenesis. Using a morphogenetic approach, has shown success in scaling gate-level EHW for a signal routing problem, however, when faced with a evolving a one-bit full adder, unforseen difficulties were encountered.

In this paper, we provide a measurement of EHW problem difficulty that takes into account the salient features of the problem, and when combined with a measure of feedback from the fitness function, we are able to estimate whether or not a given EHW problem is likely to be able to be solved successfully by our morphogenetic approach. Using these measurements we are also able to give an indication of the scalability of morphogenesis when applied to EHW.

1.1 Introduction

While evolvable hardware (EHW) has proven to be successful in the evolution of small novel circuits, generally on field programmable gate arrays (FPGAs), its applicability to complex problems has been limited, largely due to the use of direct encodings in which the chromosome directly rep-

resents the device's configuration. A practical approach to solving this problem for specific application domains has been function-level evolution, involving the use of higher-level primitives such as addition, subtraction, sine, etc. (see [3; 6] as examples). Although this scales the ability of EHW to solve more complex problems, it comes at the price of higher gate counts and designer bias [11], as well as the loss of potential novelty in solutions, thus countering some of the original motivations for EHW.

Another approach is through decomposing the problem into components or subtasks which are evolved first and then combined. Increased Complexity Evolution [10], and Bidirectional Incremental Evolution [5] are examples of this. However, these approaches are limited to applications with straightforward decompositions, without interdependencies.

A separation between genotype (the chromosome) and phenotype (the generated circuit), and a way of generating the phenotype from the genotype (a growth process), is the approach taken by nature to evolve complex organisms, and has increasingly been seen as a means of scaling EHW to more complex problems without losing its ability to generate novelty. By encoding a growth process, known as morphogenesis, rather than an explicit configuration, the complexity is moved from the genotype to the genotype-phenotype mapping.

Morphogenetic approaches have been successfully applied to generating neural networks [4; 2; 9] and tessallating patterns [1]. Recent work by the authors [7] showed that this approach could also be successfully applied to EHW at the gate level on a modern FPGA, specifically the Xilinx Virtex.

In this paper we briefly revisit the results of using morphogenesis to scale EHW on a signal routing problem, presented in Section 1.2. Then, in Section 1.3 we extend this work to evolving a one bit full adder. However, in the process of evolving adders, several obstacles were encountered, this forced us to come up with a measure of problem complexity, or more accurately *difficulty*, and fitness feedback as a means of identifying whether a given experiment is likely to succeed. This also allows us to quantify the scalability of our morphogenetic approach to EHW. This is presented in Section 1.4, after which we draw some conclusions.

1.2 Scaling with Morphogenesis

In this work biological cells correspond to functionally independent logic elements, comprised of a function generator and associated routing, within a configurable logic block (4 per CLB). Multiplexors (gates) are interpreted as proteins within a cell, with each protein representing that resource and its configuration. Inter-cellular signaling is done via shared lines, such that

for each line that connects between logic elements, each cell has a signalling protein that corresponds to the configuration of the multiplexor (mux) in the other logic element. Each cell receives a copy of the chromosome, and implements a transcription level gene expression model, such that genes are regulated by the configuration state of the logic element, and in turn generate proteins, which then reconfigure this logic element. Through the interaction of genes with the FPGA configuration and inter-cell signalling, a morphogenesis process emerges. Details of the morphogenetic EHW system implementation can be found in [8].

To test the performance and scalability of the morphogenetic EHW approach this model was used to evolve signal routing circuits with severely constrained routing that disallowed simple connection rules (see [7] for details), and was compared with a standard EHW approach using a direct encoding on a fixed-length binary chromosome.

A first set of experiments required a signal to be routed horizontally across a 5x5 CLB matrix (100 cells) from an input in the center of the west edge to an output at the center of the east edge. Then to test the scalability of each approach, the size of the matrix was increased to an 8x8 CLBs (256 cells) and the number of inputs was increased to 4, placed in the center of the West edge of the CLB matrix, and outputs was also increased to 4, spread evenly across the East edge of the CLB matrix. This required evolution to learn not just how to connect horizontally across the matrix, but also how to spread vertically from the middle outwards. Fitness, in both cases, was based on how much progress was made in routing a signal, possibly inverted, from the inputs to the outputs. The relationship between the different inputs and outputs was disregarded, it was only required that all inputs are connected and one or more of these drive the outputs. Further details of the fitness function can be found in [7].

For each set of experiments twenty evolutionary runs were done with a population size of 100 and using a steady state genetic algorithm with tournament selection without replacement. The crossover rate was set at 80%, mutation at 2%, inversion at 5%, and for the variable length chromosomes used with the morphogenetic approach, a base insert/delete rate of 0.1% was used with 50-50 chance of insertion or deletion. Each run was continued until a solution with 100% fitness was found or until stagnation (defined as no improvement in the maximum fitness attained in 1000 or 1500 generations for the first and second problem, respectively). For the morphogenesis approach, growth is done for a minimum of 30 iterations, with fitness evaluated at each step. Growth is continued if the maximum phenotype fitness for this genotype increased in the last 15 iterations, or if phenotype fitness is increasing. The genotype's fitness is given by the maximum phenotype fitness achieved during growth.

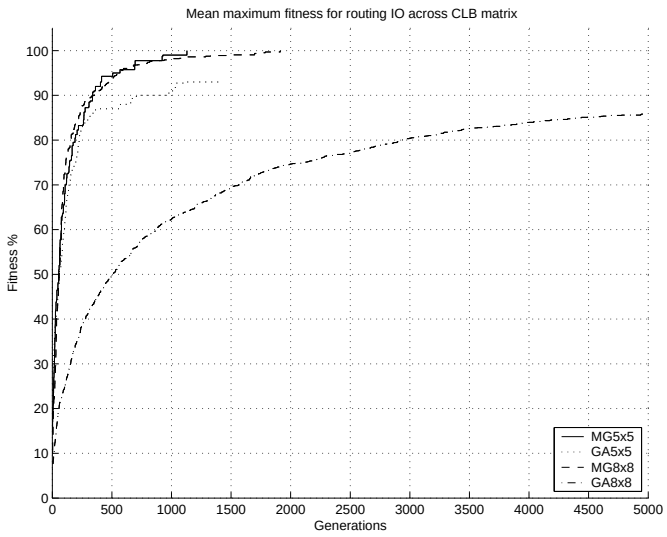

Fig. 1.1 Mean maximum fitness for routing IO across CLB matrix

Figure 1.1 show the mean maximum fitness over all runs for both approaches on the two experiments (up to generation 5000).

In the smaller problem morphogenesis approach was able to find a 100% solution every time, taking an average of 458.5 generations and 36.95 growth iterations. In comparison, the direct encoding approach was successful in only 13 of the 20 runs, with the average number of generations required for successful runs being 531.1 generations.

For the larger, more complex experiments, the morphogenetic approach was again successfully able to find a 100% solution in every run, taking an average of 1001.7 generations and 49.95 iterations. The direct encoding approach, however, was unable to solve this at all, with maximum fitness values reaching a mean of 86.6% (standard deviation was 3.1%), and taking on average 4647.1 generations. The best run, which reached 93.75% at generation 9954, was continued up to 35,000 generations, reaching a maximum of 96.875% at generation 16302.

1.3 Evolving One Bit Full Adders

While the previous section concentrated on evolving circuits with fitness primarily based on circuit structure, using a severely cut down set of gate-level resources, in this section the aim is to investigate the evolution of

circuit structure *and* functionality on a more complete set of resources. This allows us to test the scalability of our morphogenetic approach to an increase in circuit functionality, and what amounts to an increase in platform complexity.

1.3.1 *Experimental Setup*

A one bit full adder has 3 inputs (x, y, cin) and 2 outputs $(sum, cout)$, with the relation between these being defined by Boolean Equations 1.1 and 1.2.

$$sum = x \oplus y \oplus cin \tag{1.1}$$

$$\begin{aligned} cout &= x \cdot y + x \cdot cin + y \cdot cin \\ &= x \cdot y + (x + y) \cdot cin \end{aligned} \tag{1.2}$$

The target for the adder circuit is a 2x2 CLB matrix in the center of the FPGA. This size was chosen as it contains sufficient logic and routing to provide evolution with freedom to explore a wide range of possible solutions to this problem. Although an adder could be easily defined using only two 4 input function generators (LUTs) in a single slice of a CLB, most functional circuits would require a combination of LUTs with routing lines connecting them. Thus, this approach allows the results from these experiments to be more readily generalised to other functional circuits.

Cells again correspond to logic elements (there are 4 per CLB), each containing a LUT-register pair, however the register is not used here due to the combinatorial nature of the circuit. Furthermore, each cell is allocated two out buses and six single lines per out bus line, such that there are connections available to each neighbouring CLB. This is shown in Fig. 1.2, with each logic element's LUT (F or G) and connection to the out bus (for e.g. S0_Y) shown, but the connections to and from the *Out To Single* multiplexor (which is responsible for driving the logic element outputs to neighbouring cells) are aggregated for the CLB. Each LUT has available the same set of input lines as the other three LUTs in the CLB, however, each LUT has different output lines. Only directly connecting single lines between neighbouring CLBs are used (dedicated horizontally connecting out bus lines are also used, however). [1]

The evolvable region is then setup so that the x and y signals are pro-

[1]Note that 16 of the 48 single lines that we are using are able to be driven by the outputs of two neighbouring CLBs, a situation known as contention, which may damage the FPGA. To avoid this, prior to configuring an output multiplexor for driving a single line, its setting on the other end is checked, and if this configuration would cause contention, then it is not configured.

vided to single lines that can be fed to the LUT inputs in the South West CLB, while the *cin* signals are fed towards the North West CLB's LUT inputs, and the *sum* and *cout* output signals are sampled from one logic element each on the two East CLBs. This layout is shown in Figure 1.2. Notice that several lines are available to the circuit inputs, while the outputs require signals to be routed to specific lines. Signals are routed directly from IO blocks on the perimeter of the FPGA to the CLBs adjacent to the evolvable region. This approach is an artefact of the original design aims, which were to allow the evolution of asynchronous circuits for robot control.

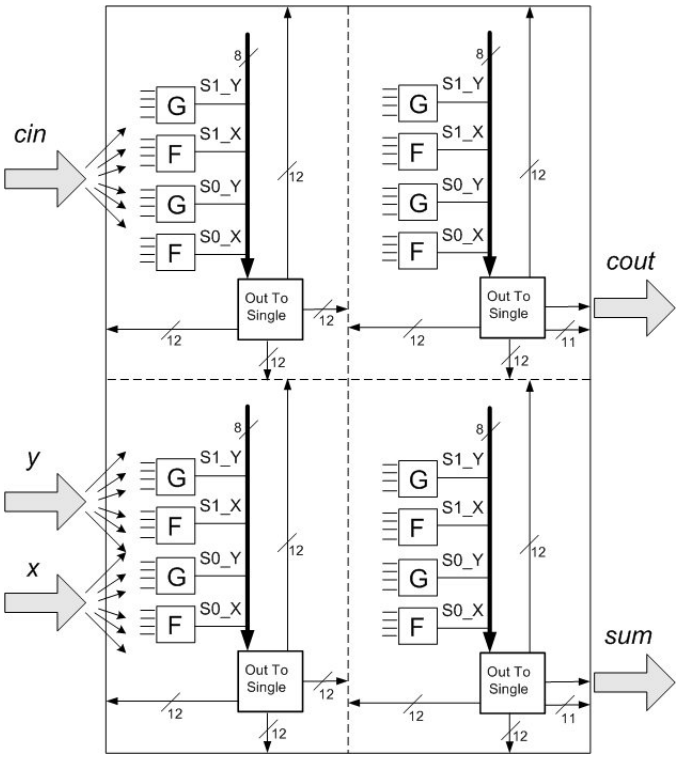

Fig. 1.2 Layout for Adder Experiments

1.3.2 *LUT Encoding*

The native encoding of the 4-input 1-output function generators (LUTs) on a Virtex is a 16-bit (2^4) inverted least significant bit first (LSB) truth

table. While this encoding, which we refer to here as LUT bit functions, is sufficient for generating any required function there may be other more evolution-friendly encodings.

One such encoding is LUT active functions, which provides basic Boolean functions (AND, OR, XOR, NAND, NOR, XNOR, Majority with tie break to 0, and Majority with tie break to 1, all with optional inversion of selected lines) that are applied only to currently *active* input lines, that being inputs that have a line connected that is driven by an active CLB output. For the line to be considered active, it must be connected to a changing signal, originating either in the circuit inputs (and generally having undergone various function transformations), or in a feedback loop, which may produce an oscillating signal (for example a clock divider).

This is based on the idea that cell functionality should only use inputs that are active. In other words, unconnected inputs and undriven input lines, that are held high (i.e. at logic 1), should be eliminated from LUT functions, where they may possibly dominate the function applied to the other inputs. An example of this would be a simple OR function of all LUT inputs, which if any one of these input lines is disconnected, then the output from the LUT will always be a 1, no matter what signals are received on the other lines. In nature, it is unlikely that this would occur (for example in neurons or signalling pathways), as it would be undesirable for biological functionality to be crippled by a large state space of useless functions, especially when there are a large number of possible inputs (for example dendrites), which would often be inactive. In EHW, however, we are utilising LUTs which are designed for human or automated design software, such that they allow a great deal of flexibility in expressing Boolean functions within the constraints of an FPGA.

1.3.3 *Fitness Evaluation*

Fitness is based on how much progress is made in connecting from the inputs to the outputs, *and* on how closely the connected outputs match that of the desired function. Both connectivity (c) and functional adder fitness (a) components are given as a percentage (i.e. in the range 0 to 100 inclusive). The circuit fitness (f) is given by adding these together and then scaling the result back to a percentage (i.e. from 0 to 100), hence

$$f = (c + a)/2. \tag{1.3}$$

The circuit connectivity portion of fitness is produced in one of two ways, depending on which of the LUT encoding methods is used. If LUT active functions are used, then a recursive connectivity test is done based on the FPGA's current configuration. Fitness is calculated in the same

manner as was done for the previous signal routing experiments. Briefly, connectivity fitness is calculated by testing how many layers are connected, and for each connected layer, how many elements in the layer are connected.

For active functions, there are 5 levels of connectivity measurable within a logic element. If LUT active functions aren't used (native encoding), then each logic element is only assigned a connectivity value based on whether or not there are signal changes (indicating connectivity) on the probe at the slice output, when signals are applied to the circuit input bus. Thus, for LUT bit functions, there are only 2 levels of connectivity measurable within a logic element.

The adder function fitness component is evaluated only when one or more circuit outputs are connected. To evaluate the circuit's functional fitness, each of the input signal combinations in the adder's truth table is iterated through and the connected circuit outputs are tested to see how closely they match the desired corresponding entry in the truth table.

Function equivalence is measured according to how close the circuit output matches the truth table's output signals, based on the Hamming distance between the outputs that *change* and the specified outputs, while unchanging outputs are assumed to be unconnected to the inputs, and for each of these 1 is added to the Hamming distance. Adder function fitness is then based on the proportion of matching signals, calculated as

$$a = 100(1 - \frac{h}{l}) \tag{1.4}$$

where h is the Hamming distance (with an integer value from 0 to l), and l is given by

$$l = On_c = 2n_c \tag{1.5}$$

with O being the width of the circuit output bus (i.e. the number of outputs from the adder function), and n_c is the number of input signal combinations required to specify each output's truth table, as given by

$$n_c = 2^I = 2^3 = 8 \tag{1.6}$$

where I is the width of the circuit input bus (i.e. the number of inputs to the adder function).

1.3.4 *Experiment Results*

Two evolutionary runs were done for both LUT encodings (time limitations prevented more than this), each using the same evolutionary and morphogenetic parameters as used in the previous signal routing experiments, with

evolution being stopped at 5000 generations or after stagnation for 2000 generations. The results of all the runs is given in Fig. 1.3.

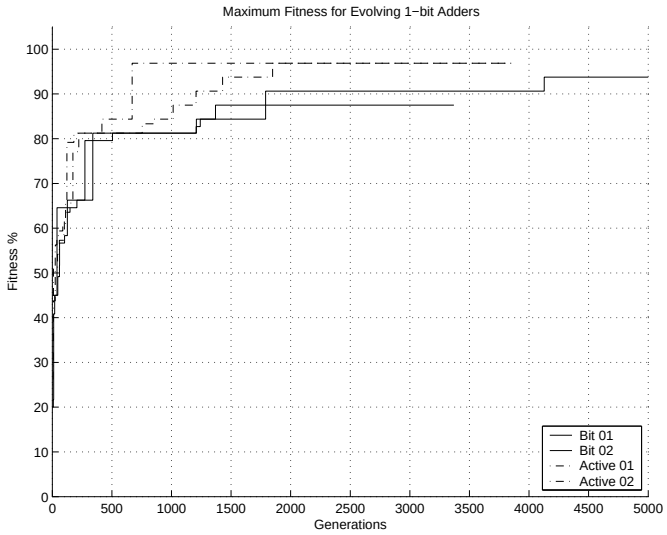

Fig. 1.3 Maximum Fitness for Evolving 1-bit Adders

Unfortunately, no approach was able to find a 100% solution, the best achieved was 96.875% (which means only 1 of the 16 output signals was incorrect), which was achieved by both active LUT runs (at generations 670 and 1849). The native (LUT bit function) encodings were only able to achieve fitnesses of 93.75% and 87.5% (corresponding to 2 and 4 wrong signals) at generations 4128 and 1369, respectively. Although these runs are not statistically significant, they can be seen as a rough indication of trends for the different approaches.

LUT active functions, not surprisingly, were slower off the mark at solving the problem, as each LUT is only able to encode a single simple Boolean function, requiring more LUTs to be configured and the routing between them to be developed, before they are able to offer any advantage. However, it is obvious that this is what occurred, and once this had been established, it rapidly overtook the other approach.

1.3.5 *Further Experiments*

To investigate where evolution fails in its task of creating a 1-bit full adder, we divided the problem into two separate components: circuit function, and

circuit routing. To do this we ran two separate sets of experiments. In the first, we pre-connect the functional blocks together (i.e. fix the routing lines and the LUT inputs) and evolve the LUT functions, while in the second, we fix the LUT functions and evolve the routing between them.

Two evolutionary runs were done for both LUT encodings (again time limitations prevented more than this), each using the same evolutionary and morphogenetic parameters as before, with evolution being stopped at success or after 1000 generations of stagnation. The results of these are given in Table 1.1.

Table 1.1 Adder Fixed Run Results

Approach	Run 1		Run 2	
	Max Fitness	At Gen	Max Fitness	At Gen
BitFN fixed Mux	100.0	214	100.0	434
BitFN fixed LUT	91.25	1514	93.75	1417
ActiveFN fixed Mux	100.0	71	100.0	56
ActiveFN fixed LUT	94.8	728	90.625	993

This clearly shows that the problem lies in the evolution of the routing between LUTs, while LUT functions are able to be evolved quite rapidly.

1.4 Analysing Problem Difficulty

Here we analyse the difficulty of the various experiments, with the aim of developing a heuristic that will help us to determine whether or not any given experiment is likely to be solvable by evolution.

To do this, we need some measure of difficulty and of the feedback provided by the system to guide evolution towards a solution. The first measure is the state space of the problem, calculated as the total number of configuration states that the EHW system may configure the FPGA to. This is calculated for a logic element as the product of the number of settings (σ) for each of the m multiplexors:

$$M = \prod_{i=1}^{m} \sigma_i \tag{1.7}$$

then the number of configuration states is given as

$$S = M^n \tag{1.8}$$

where n is the number of logic elements being evolved (this could also be done per CLB).

The next measure is the number of solutions, or answers A, to the problem available within this configuration space. This is problem dependent, but in all cases, elements of the configuration space that don't affect the solution (i.e. redundancies), increase the number of solutions, as

$$A = \alpha M^{\rho} \tag{1.9}$$

where α is the initially calculated number of solutions for the essential (or actively participating) circuit components (logic elements), and ρ is the number of redundant logic elements.

With the state space (S) and number of solutions (A) calculated, then the probability (P) of a randomly generated configuration being a solution is given by

$$P = A/S \tag{1.10}$$

which can be seen as a measure of problem difficulty.

We also need to measure the size of the fitness feedback space, this being the amount of information provided by the system to the fitness function for guiding evolution. This can be calculated in a general manner using the amount of feedback from the circuit elements, given per logic element as μ, and overall function, given in terms of the number of circuit inputs (I) and outputs (O) required to completely describe the function. The total fitness feedback space is then given as

$$F = \mu^{n} 2^{(2^{I} O)} \tag{1.11}$$

where the 2's in the (latter) overall circuit function component of the equation are due to the binary nature of digital signals.

However, while this tells us the total amount of information available to the fitness function, it doesn't take into account how much of this information is actually used by the fitness function. For example, in our experiments, when we measure connectivity not all logic elements' feedback is utilised. In this case, in the circuit input and output layers (columns of CLBs) we only look at the connectivity of the circuit input and output logic elements, while all others in the layer have their connectivity measure dicarded in the fitness evaluation, reducing n to n'. This would reduce the above measure of F to an *effective* fitness feedback space (E) of

$$E = \mu^{n'} 2^{(2^{I} O)} \tag{1.12}$$

1.4.1 *Experiment Difficulty Comparisons*

With the measures introduced above, we are now able to analyse and compare the measures of difficulty for each set of experiments that were conducted. Due to space limitations, only the results of the calculations for the various measures are provided here.

It should be noted that as the number of solutions is increased by the amount of redundancy created by unused logic elements (that are able to have any configuration without affecting the circuit's function). This means that for the signal routing problems, only the shortest path needs to be found to give a first order approximation to the number of solutions (including the longer paths would have no affect on the calculated values here, however, due to their greatly reduced order of magnitude).

Note also, that for the unconstrained adder problem, where we evolve both the multiplexor settings and LUT functions together, an over approximation was used for the number of solutions. This was calculated starting with the number of solutions from multiplexor settings only evolution, where the LUT functions were fixed, and noting that for each funtion that is needed to generate an adder, there are several possible LUT functions that are able to achieve this, so the initial estimate is multiplied by the number of solutions from LUT only evolution. In other words the number of possible wiring connection settings for one particular valid set of adder function blocks is multiplied by the number of possible LUT function combinations for one of these particular wirings. Then, to take into account different routing and function combinations that could also work, this last estimate is then multiplied by the number of logic elements (n) that are being evolved. This gives

$$A = A_{mux} \cdot A_{lut} \cdot n \tag{1.13}$$

which is of course an over estimate, and not at all accurate, but is simply used to give some sort of estimate as to the upper limit of the order of magnitude.

For LUT active functions, we also need to take into account the fact that the other LUTs that weren't used with fixed routing (they are inactive) could take various configurations usually without upsetting the circuit's function, so n is squared for good measure, to give

$$A = A_{mux} \cdot A_{lut} \cdot n^2 \tag{1.14}$$

A comparison of the measures for each experiment set are given in Table 1.2, noting that $\sim$ indicates an approximation using an average for variables, and $<$ is estimated over approximation only. For the experiment names, *_lut* indicates evolving LUT functions (fixed mux), *_mux* indicates

Table 1.2 Experiment Difficulty Measures

Experiment	S	A	P	E	E^*			$-P/E$	Gens
Adder_lut_act	38	17	−21	22	6	–	18	0.95	6
Adder_lut_bit	256	231	−25	22	7	–	22	1.14	8
Adder_mux_act	464	334	−130	28	37	–	114	4.64	-
Adder_mux_bit	464	385	−79	22	37	–	114	3.59	-
Adder_act	~ 544	<375	< -169	30	48	–	149	5.63	-
Adder_bit	720	<623	< -97	22	28	–	85	4.41	-
5x5:1-1	999	963	−36	144	10	–	32	0.25	9
8x8:4-4	2558	2351	−207	464	59	–	182	0.45	10

evolving mux settings (fixed LUT), *_act* indicates LUT active functions only used, *_bit* indicates LUT bit encoding only used, and *5x5:1-1* and *8x8:4-4* are used to denote the signal routing experiments (with generations taken for the morphogenetic approach only). Only experiments that were able to find a 100% solution are given entries for the *Gens* (average generations taken) column. Note that all entries are base 2 logarithms, rounded to the nearest integer.

From this it can be seen that the actual increase in problem difficulty between the smaller and larger routing problems was

$$P_{5x5} - P_{8x8} = 2^{-36+207} = 2^{171} \approx 3 \times 10^{51} \tag{1.15}$$

This shows that our morphogenetic approach is able to scale to a problem of more than 50 orders of magnitude greater difficulty, with only an increase of around 3 times more work (measured by the proportional increase in generations and growth iterations), while the standard direct encoding approach to EHW struggled (65% success rate) even with the simpler problem and totally failed to scale to the more difficult problem. Notice that, the morphogenetic approach has the desirable feature that, as P decreases (indicating an increase in problem difficulty), the number of generations required to solve the problem increases very slowly, in fact in a better than $log_2 N$ manner.

By examining this table, it seems that $-P/E$ correlates well with the solvability of the problem, by a *morphogenetic* approach to EHW. Tentatively, we can say that for the problem to be solvable this needs to be less than or equal to k, where from experimental evidence $1.136 \leq k < 3.593$ ($25/22 \leq k < 79/22$). That is

$$-log_2(P)/log_2(E) \leq k \tag{1.16}$$

which can be expanded (noting that $P^{-1} = S/A$) and rearranged to give

$$log_2(E) \geq log_2((S/A)^{1/k}) \tag{1.17}$$

and as log_2 is a monotonic function, we can remove it from both sides while preserving the relation, thus

$$E \geq (S/A)^{1/k} \tag{1.18}$$

We can denote $(S/A)^{1/k}$ as E^*, which tells us the minimum amount of effective feedback required to successfully guide the morphogenetic system to a solution. Hence we require that $E \geq E^*$ for a problem to be solvable.

The range of values of E^*, using best (3.5) and worst (1.136) cases for k's value, for each experiment are given in Table 1.2. As can be seen, the routing experiments, while they are harder to solve, according to P, remain solvable as they have a more than sufficient amount of effective feedback ($E > E^*$) available to guide evolution. while, on the other hand, the unfixed adder problem and the adder problem where the multiplexor settings need to be evolved both lack sufficient feedback ($E < E^*$) to guide evolution to a complete solution. It can also be seen that evolving the adder LUTs was successful due to there being a larger proportional coverage of feedback to the problem, ensuring that there was sufficient feedback ($E \geq E^*$).

1.5 Conclusion

In this paper we have shown that morphogenesis scales extremely well to increases in circuit size and problem difficulty. This offers great promise to EHW, as it provides scalability without having to compromise the advantages of gate-level evolution.

We have also introduced a quantitative measure of problem difficulty, in terms of the probability of finding a solution, and a heuristic indicating whether the problem is solvable or not according to its difficulty and the amount of information provided by the fitness function for guiding evolution. This shows the importance of fitness feedback, and to a lesser degree problem difficulty, to a morphogenetic approach, while the state space size, reflecting FPGA platform complexity and circuit size, has little direct impact.

Ideally speaking, a problem should be analysed prior to applying evolution to solving it. If, according to the heuristics provided, it appears unsolvable, then either more feedback is required, or the probability of finding a solution should be increased, by decreasing the search space or by introducing more redundancies that effectively do the same.

Using morphogenesis coupled with these heuristics (refined through further experimentation), gate-level EHW remains a viable method and should continue to be able to provide novel solutions to hardware design problems.

Bibliography

Peter Bentley and Sanjeev Kumar. Three ways to grow designs: A comparison of evolved embryogenies for a design problem. In *Proc. of the Genetic and Evolutionary Conference (GECCO '99)*, pages 35–43, 1999.

Peter Eggenberger. Cell interactions as a control tool of developmental processes for evolutionary robotics. In *Proceedings of SAB '96*, pages 440–448, 1996.

Tetsuya Higuchi et al. Evolvable hardware at function level. In *IEEE International Conference on Evolutionary Computation*, pages 187–192, 1997.

Nick Jakobi. Harnessing morphogenesis. Technical Report CSRP 423, School of Cognitive and Computer Science, University of Sussex, 1995.

Tatiana Kalganova. Bidirectional incremental evolution in extrinsic evolvable hardware. In *The Second NASA/DoD Workshop on Evolvable Hardware*, pages 65–74. IEEE Computer Society, 2000.

Tatiana Kalganova. An extrinsic function-level evolvable hardware approach. In *Proc. of the Third European Conference on Genetic Programming (EUROGP2000), LNCS 1802*, pages 60–75. Springer-Verlag, 2000.

Justin Lee and Joaquin Sitte. A gate-level model for morphogenetic evolvable hardware. In *Proc. of the 2004 IEEE International Conference on Field-Programmable Technology (FPT'04)*, pages 113–119, 2004.

Justin Lee and Joaquin Sitte. An implementation of a morphogenetic evolvable hardware system. In *5th International Conference on Simulated Evolution And Learning (SEAL04)*, Busan, Korea, 2004. Korea Advanced Institute of Science and Technology (KAIST).

Daniel Roggen, Dario Floreano, and Claudio Mattiussi. A morphogenetic evolutionary system: Phylogenesis of the poetic circuit. In *Proc. of the 5th International Conference on Evolvable Systems: From Biology to Hardware ICES 2003, LNCS 2606*, pages 153–164. Springer, 2003.

Jim Torresen. A divide-and-conquer approach to evolvable hardware. In *2nd International Conference on Evolvable Systems: from biology to hardware (ICES 98), LNCS 1478*, pages 57–65. Springer-Verlag, 1998.

Vesselin K. Vassilev and Julian F. Miller. Scalability problems of digital circuit evolution: Evolvability and efficient designs. In *Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware*. IEEE, 2000.