The Input Pattern Order Problem II: Evolution of Multiple-Output Circuits in Hardware

Martin A. Trefzer, Tüze Kuyucu, Julian F. Miller and Andy M. Tyrrell

Abstract— **It has been shown in previous work that the design of the input pattern plays and important role in intrinsic evolvable hardware. The results have shown that randomising the input pattern during the course of evolution is mandatory in order to achieve valid circuits that work correctly under all conditions. Furthermore, it was found that randomising the input pattern order helps evolution to avoid local optima. Due to the fact that it becomes exponentially more difficult to successfully evolve circuits with increasing number of outputs, this paper addresses the importance of the input pattern order problem (IPOP) for the second time, with a focus on the intrinsic evolution of digital circuits with multiple outputs. In this paper, we investigate whether it is also possible to design the input pattern in a way to improve the performance of evolution when tackling multiple output problems. A different approach to IPOP is taken, where input pattern order is used as a method of balancing the distribution of the output values. The aim is to equalise the fitness value range by using weighted input vectors in order to obtain a smoother search landscape for multiple output circuits. The proposed method is tested on the intrinsic evolution of 4-bit AND, 4-bit parity, 2-bit full adder, and 2-bit multiplier. Furthermore, the results for the intrinsically evolved multipliers are compared with software simulations in order to determine whether there is a benefit for both intrinsic and extrinsic evolution.**

I. Introduction

Scalability is a major challenge in Evolvable Hardware (EHW). Various systems and techniques have been developed and are being developed to aid evolution in achieving circuit designs of high complexity [3], [19], [21]. For both intrinsic and extrinsic evolution of electronic systems, successfully evolving circuits with multiple outputs is particularly hard. As the number of inputs and outputs (IO) increases, the difficulty level of given tasks increases exponentially [14], since additional dimensions are added to the search space. Hence, the evolution of circuits with a high IO count becomes considerably more complex. This is especially true for intrinsic evolution because of the constraints that are imposed by the respective hardware architecture.

There are two main approaches that use partitioning to overcome the problems with evolving multiple output circuits: in the first case the truth table is decomposed into smaller sub-problems, which are then merged together again to obtain the final circuit (incremental evolution [8], [14] and input vector partitioning [17]). While the first approaches rather focus on the inputs, the second approach is to evolve sub-circuits separately for each output. This technique is successfully used for the extrinsic evolution of large multiple output circuits [20]. It is interesting to see that the largest logic circuits evolved between 1992 and the present have always been single output circuits (parity, tonediscriminator circuits) or multiple circuits evolved with an output partitioning approach; so that each output is effectively evolved as a separate, one output circuit [9], [10], [13], [15]–[17], [20].

Another important issue in evolvable hardware is the evolution of reliable circuits. Evolved circuits can only be guaranteed to always function correctly, or at least be considered to be more reliable—rather than merely producing intermittent solutions— when the input pattern order is randomised during the evolution process [7], [18]. Evolution is particularly good at finding specific solutions, which are only valid for exactly the input pattern order that has been applied during the evolution experiment. Thus, an evolved circuit might well be just a pattern generator that always generates the desired output irrespective of the actual applied inputs. In [18], it has been shown that using randomised input pattern order has two benefits: first, it helps evolution to find valid circuits and second, it decreases the likelihood of getting stuck in local optima.

A set of input patterns consists of the finite number of entries of a truth table, which all have to be included into the test pattern in order to entirely define the desired digital circuit. Input pattern order problem (IPOP), refers to the arrangement of patterns of the desired problem inputs in their application to the circuit in training.

Considering the importance of the IPOP for the results achieved in the earlier work [18], the question arises whether the input pattern can also be shaped in a certain way to improve the performance of evolution when tackling circuits with multiple outputs. Therefore, in this paper we address the IPOP on the intrinsic and extrinsic evolution of logic circuits for the second time: a method of balancing the target output, and therefore the fitness value range for each input vector, by structuring the input pattern order is proposed. The introduced method uses weighted input patterns in order to obtain a smoother fitness landscape for multiple output circuits.

Results from different experiments are presented to show the effects of the unbalanced outputs on the performance of evolution, especially in the case of intrinsic evolution. As in previous work, the presented intrinsic experiments are carried out on the reconfigurable integrated system array (RISA) evolvable hardware platform [4]. The experiments presented include; 4-bit AND, 4-bit even parity, 2-bit adder and 2-

Martin A. Trefzer, Tüze Kuyucu, Julian F. Miller and Andy M. Tyrrell are with the Department of Electronics, Intelligent Systems Group, University of York. {mt540, tk519, jfm7, amt}@ohm.york.ac.uk, http://www.bioinspired.com.

This work is part of a project that is funded by EPSRC - EP/E028381/1.

bit multiplier in hardware. The multiplier, being the circuit with the highest output count, is considered as the hardest to evolve. It is further investigated whether balancing the outputs also affects the outcome of extrinsic evolution experiments. Results for 2-bit multipliers using CGP are compared with the results obtained using hardware. The comparison of the extrinsic and the intrinsic case is rather intended to reveal whether balancing the outputs is a beneficial concept for both hardware setups where feedback and delays matter and a pure combinatorial logic setup in software; rather than to compare the performance of the approach on different platforms.

II. Challenges in the Evaluation Process in EHW

One of the most important steps in evolutionary experiments is the evaluation process: candidate solutions are tested and rated by a fitness value, which determines how successful they are in solving a given task. It is the evaluation process that provides the foundation for the selection process, and thereby greatly contributes to guiding evolutionary search through the solution space in order to construct the desired circuits. It was shown earlier that a fitness function, which is more suited for the particular problem at hand had a considerably positive impact on the performance of evolution in designing the desired circuit [11], [18]. The importance of different selection schemes, being the second major driving force in evolutionary algorithms, have also been addressed by various researchers [2], [22].

Although evolution is guided to the same extent by both the fitness function and the selection process, of the limiting factors for these processes often emerges from the information available from the evaluation stage: the inputs applied and the outputs measured during the testing process determine the shape and size of the search space evolution can sample.

Therefore, it is important that the evaluation process is provided with as much and as accurate information as possible in order to provide precise and useful information to the selection process. In the evolution of circuits in hardware, the main (and most of the time the only) available inputs to the evolved systems are the inputs from the truth table that define the problem at hand. The number of times and the pattern of the inputs applied can therefore be used to shape the search space sampled by the EA. In fact the IPOP can be used to tackle various important challenges in EHW: avoiding local optima, achieving fully functional solutions and improving the performance of evolution.

A. Getting Stuck in Local Optima

The most likely reasons for getting stuck in local optima during the course of evolutionary search are: first, unsuitable fitness evaluation methods that are not capable of considering special cases within certain problem domains [11]. Second, intermittent solutions that can mislead the search process and cause the EA to stall. Third, rugged fitness landscapes can impede the EA in finding the optimal solution. The shape of the fitness landscape is determined by the assessment of the output values and the genetic representation as well as the problem. As the given problem itself cannot be changed, the fitness evaluation method and the genetic representation remain to be optimised. For instance, an approach where the genetic representation is dynamically changed during evolution by applying different grey codings can be found in [1].

TABLE I

Truth table for a 2-bit multiplier and a 4-bit Parity. For the multiplier, the number of input combinations that return an output of 0 are the most common, where the outputs 1, 4, 9 occur for only one input combination. When a 2 bit multiplier is evolved by testing the circuits using all input combinations the same number of times in every evaluation step, more emphasis is given in obtaining outputs that equal 0 than any other, and little emphasis is given in obtaining the correct outputs for the cases where the outputs are 1, 4 and 9; even though achieving the correct result for any input combination should be equally important for a fully functioning circuit. Unlike the multiplier, the parity problem has even number of input combinations that return the outputs 1 and 0. Thus there is equal pressure in achieving either of the output values, and the search process is not biased to satisfy some outputs earlier (more likely) than others.

A_0	A_1	B_0	B_1	Parity	Multiplier
0	0	0	0	0	0000 = 0
0	0	0	1	1	0000 = 0
0	0	1	0	0	0000 = 0
0	0	1	1	1	0000 = 0
0	1	0	0	0	0000 = 0
0	1	0	1	1	0001 = 1
0	1	1	0	0	0010 = 2
0	1	1	1	1	0011 = 3
1	0	0	0	0	0000 = 0
1	0	0	1	1	0010 = 2
1	0	1	0	0	0100 = 4
1	0	1	1	1	0110 = 6
1	1	0	0	0	0000 = 0
1	1	0	1	1	0011 = 3
1	1	1	0	0	0110 = 6
1	1	1	1	1	1001 = 9

As mentioned before, including randomness in the input pattern order and using suitable fitness functions account for the first two reasons for getting trapped in local optima. To a certain extent, randomising the input can also indirectly help in the case of rugged fitness landscapes, due to the fact that evolution is less likely to converge into local optima that are only stable when a certain input pattern order is applied. This issue can further be tackled by the IPOP, which is addressed in detail in Section III of

B. Avoiding Intermittent Solutions

When provided with a rich substrate, e.g. a substrate that features sequential logic and allows feedback loops to occur, evolution is likely to find solutions that can be termed as 'intermittent' solutions, when constructing a circuit. Intermittent solutions refer to the circuits that do not satisfy the desired circuit characteristics under all circumstances. If static or

periodic input patterns are used to find solutions for circuits, it will be likely that evolution finds circuits that produce the desired output; the output would not be correlated to the input and the circuit therefore will generally fail for random test patterns. Applying randomness in the order of applied input patterns is possibly one of the most crucial things that have to be ensured when setting up evolution experiments that depend on those patterns.

By creating a new random input pattern for each generation prevents evolution to exploit regularities in the input pattern and therefore drives evolution to find more general solutions to a given task. It was shown earlier that it is crucial to include randomness in the input patterns applied during the course of evolution in hardware, in order to obtain solutions that are able to cope with any order or sequence of the inputs [18].

C. Improving Effectiveness of Evolutionary Search

Improving effectiveness of the evolutionary search refers to efficiently sampling the fitness landscape. An approach to tackle this via resolving unbalanced output value distributions of given problems is introduced in this paper.

A good example of a problem with unbalanced output value distribution would be a 2-bit multiplier, explained in Table I. An example for a problem that inherently features balanced output distribution would be a parity circuit where there are even numbers of 1s and 0s as outputs, see Table I. Thus, the frequency of which the outputs occur is not driving evolution towards local optima.

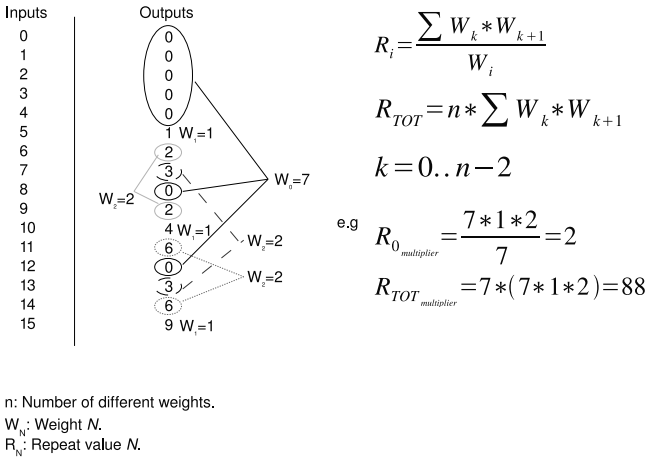
In the case of a problem with unbalanced outputs, evaluating the circuit using the input pattern order unaltered from the truth table will create an uneven search space, with local optima around the outputs with high number of occurrences. Especially when done intrinsically in hardware, due to many limiting properties of hardware substrates like routing and fixed I/O locations, evolution can easily get stuck at these local optima thus failing to successfully evolve the desired circuit. The cost of getting out of this kind of local optima is so high that it becomes almost impossible to get out again.

A simple and generic technique on balancing the input patterns will be introduced in Section III.

III. Guiding Evolution Out of Local Optima via Balancing Input Patterns

As it was mentioned in Section II-C, circuits with unbalanced outputs—where a certain (combination of) output(s) occur more (or less) than others—create local optima when being evolved especially in hardware, often trapping evolution. Examples of such circuits include multipliers and adders. An input pattern that produces every possible output value equally as often (i.e balanced outputs) is referred to as (output) balancing input pattern.

Balancing of the outputs is achieved by assigning a weight to each output combination, which corresponds to the number of times it appears in the truth table; the more often it appears the greater is the weight. Using these weights the input pattern applied during the evaluation process can be shaped in a way that combinations of inputs that produce less frequently occurring outputs (outputs with smaller weights) are repeated more often. Thus, balancing the number of occurrence of each output combination is achieved.of



$$R_i = \frac{\sum W_k * W_{k+1}}{W_i}$$

$$R_{TOT} = n * \sum W_k * W_{k+1}$$

$$k = 0..n-2$$

$$R_{0_{multiplier}} = \frac{7*1*2}{7} = 2$$

$$R_{TOT_{multiplier}} = 7*(7*1*2) = 88$$

Fig. 1. An example process determining the weights and repetition parameters for the inputs and outputs for the 2-bit multiplier problem is demonstrated. Note that multiples of R_{TOT} are ideal input pattern sizes that need to be applied to achieve a perfectly balanced output pattern. The greater the weight (W) of an output value, the more biased evolution will be towards that output, if output balancing input patterns are not used.

An example of how the weights are determined in the case of the 2-bit multiplier is shown in Figure 1. In order to obtain the desired input pattern that yields balanced outputs, the weights can be used to obtain the number of repetitions for each input combination. Note that the number of input samples is set to 128 for all experiments in this paper. Although for experiments like the full adder, a much larger number of input samples would be required to obtain a perfectly balancing input pattern, 128 is used due to memory limitations in the current hardware system. This means that also for the multiplier example where a sample size of multiples of only 88 are required to represent a perfectly balancing input pattern, the number of samples used is set to 128 for simplicity purposes. As a consequence, in the experiments presented where balancing input patterns are used, the balancing of the outputs are actually not perfectly achieved. The corresponding number of occurrences of each output in the unbalanced and the balanced case respectively are shown in Figure 2.

Using an input pattern that yields balanced outputs during evolution to evaluate the candidate circuits provides a smoother search space for the evolutionary algorithm, since there is no longer a bias towards solutions that only satisfy a more frequently sampled subset of the output value range. Hence, reducing the number of local optima. Experiments are presented in Section V to demonstrate the effects of output balancing input patterns in the evolution of circuits in hardware.

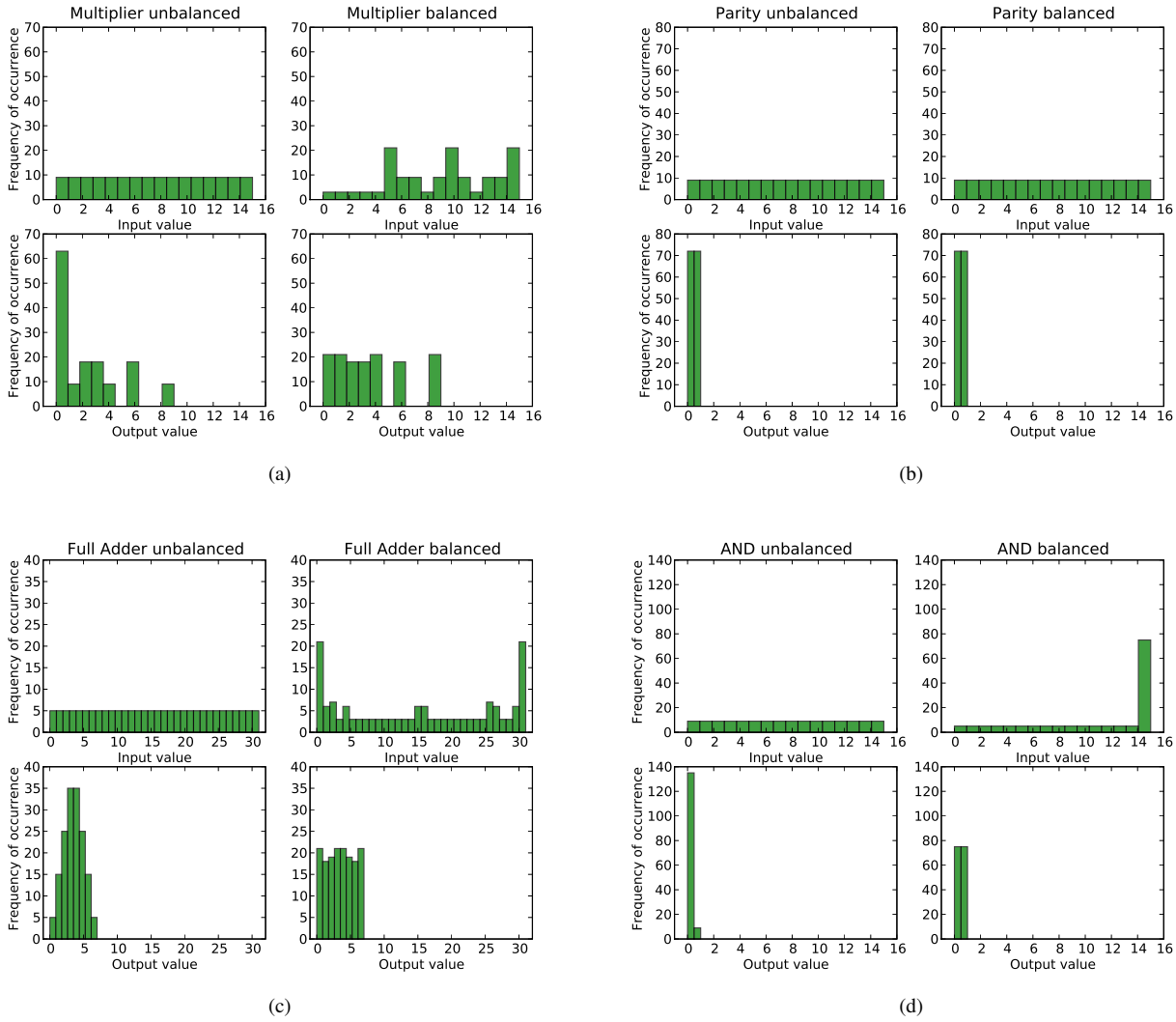


Fig. 2. The corresponding output occurrences are shown for the balanced and unbalanced output patterns used in the experiments presented in this paper for all the circuits. In the case of the multiplier and the AND, it should be obvious that with unbalanced outputs, the output value 0 is favoured more than any other output, which would encourage evolution to prefer circuits that mainly produce 0s for their outputs: a highly undesirable bias for intrinsic evolution where a stuck at 0 could achieve a higher fitness score. On the other hand, for the parity circuit the non-balancing and the balancing input pattern orders are identical.

IV. Experimental Setup

Two different kinds of experiments are carried out, in order to obtain the results presented: intrinsic experiments using reconfigurable integrated system array (RISA) [5], and extrinsic experiments using cartesian genetic programming (CGP), a widely used extrinsic approach to the evolution of digital circuits, introduced in [12].of

A. Evolvable Hardware Platform

Intrinsic evolution experiments are carried out using an embedded system hardware setup, which consists of the RISA chip and a Xilinx Spartan3E FPGA. The actual evolvable hardware platform is RISA, a custom made reconfigurable digital device, which was designed at the Department of Electronics, University of York [5]. Spartan3E FPGA is used to host the evolutionary algorithm, and operate and interface RISA.

RISA consists of an array of 6×6 clusters, surrounded by input/output (IO) cells, as shown in Figure 3. The IO cells provide a total of 12 IOs at each side of the RISA module (each cell providing 2 IOs), which can be independently configured as either an input or an output. Each cluster provides four functional units that comprise a 16 bit look-up table (LUT), a 3 bit multiplexer (MUX), and a flip-flop. These functional units, the available routing resources and the possibility of creating feedback loops offer a rich variety of configuration options to the EA.

In addition, the FPGA offers features that make it particularly suitable for evolution experiments: first, it is designed in a way that it cannot be destroyed by random bit strings. The latter feature is not generally present in current commer-

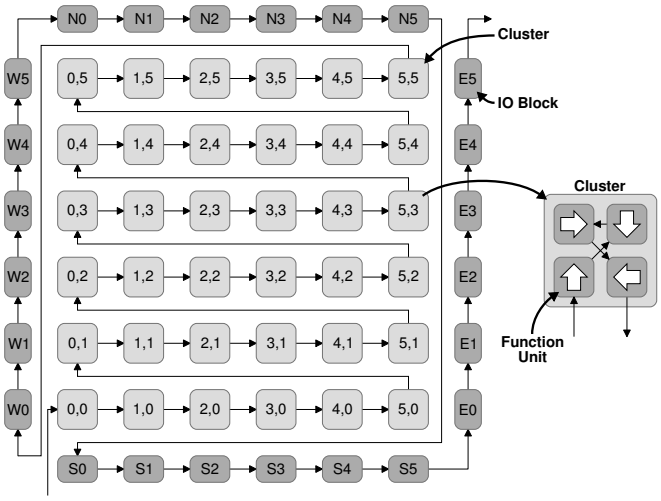


Fig. 3. The FPGA substrate of RISA consists of an array of 36 functional clusters surrounded by input/output (IO) blocks. Each cluster and IO block can be configured individually, providing partial reconfiguration. The clusters offer a rich variety of routing and logic configuration options.

TABLE II
EVOLUTIONARY STRATEGIES PARAMETERS FOR BOTH THE SOFTWARE AND HARDWARE EXPERIMENTS.

Parameter	EA Hardware	EA Software
Parent Size	2	2
Population Size	7	7
Generation Limit	15000	15000
Mutation Rate	0.5...10%	1...25%
Number of Runs	20	100
Genome Size	4608 bits	100 nodes (3200 bits)

cial FPGAs: the synthesis tools of the manufacturers either constrain the access to the bit-string, in order to protect the device, or it is actually possible to destroy it. Second, the configuration of clusters can be changed independently from each other, hence, the logic offers partial reconfiguration. This can considerably accelerate hardware evolution [6], since only those parts of the bit-string, which have actually been changed by the EA, need to be reloaded into the device, instead of reconfiguring the entire device.

B. Evolutionary Algorithm & Genotype

An evolutionary strategy (ES) (2+5) is used for the experiments that are undertaken. The mutation rate is adaptive and encoded in the genome; prior to the mutation operation, the current mutation rate is increased, decreased, or kept by a normally distributed random number between -1 and 1. Subsequently, mutation is carried out probabilistically with the new mutation rate. Evolution is stopped when either the solution is found, or the generation limit of 15,000 is reached. A total of 20 randomly initialised evolution runs have been carried out for all hardware experiments and a total of 100 independent runs are performed in the case of evolution in software. The genotype consists of 128 configuration bits per RISA cluster, resulting in a total of $36 \times 128 = 4608$ configuration bits to configure the RISA FPGA. See Table II for the list of the evolutionary algorithm parameters.

V. BALANCED VS UNBALANCED OUTPUTS IN HARDWARE AND SOFTWARE CIRCUIT EVOLUTION

The experiments that are undertaken to investigate the effectiveness of the balancing input pattern order are presented in Table III. Different problems are tackled in the case of intrinsic evolution: 4-bit AND, 4-bit parity, 2-bit full adder and 2-bit multiplier. A total of 20 independent runs are carried out for each task. It can be seen from Table III that solutions to 4-bit parity and 4-bit AND are found quickly in the evolution runs. This is due to the fact that both problems are relatively simple circuits with only one output. However, the results show that even though evolution reliably finds solutions for simple problems like a 4-bit AND, a speedup of factor two can still be observed when output balancing input patterns are used. The 4-bit parity is included as an example where the output value distribution is already balanced. This suggests that parity—on its own—is not a suitable test problem for this kind of evolution experiments, as the impact of the presented approach cannot be determined.

The intrinsic evolution of 2-bit full adders and 2-bit multipliers are non-trivial tasks. As can be seen from the graphs in Figure 2, these functions feature not only multiple outputs but also a greatly unbalanced output value distribution. Thus, it is not surprising that the success rate of evolution is low in the cases where output balancing is not applied; for the 2-bit full adder only two and for the 2-bit multiplier only three out of twenty runs are successful. However, it can be observed that the success rate was increased by a factor of four for the 2-bit full adder, and a factor of three for the 2-bit multiplier in the runs where the balanced outputs method is applied.

In order to investigate whether the benefit that comes with balancing the output value distribution is restricted to intrinsic circuit evolution or it is also advantageous in the extrinsic case, additional experiments are carried out for the 2-bit multiplier in CGP. Since CGP is fairly quick in evolving 2-bit multipliers, compared to an embedded system, it is feasible to perform 100 evolution runs. Although the benefit is not as significant in software as it is in hardware, a speedup in the region of 20% can also be observed for extrinsic evolution experiments. The fact that extrinsic results are not as much affected by the unbalanced output distributions, i.e. the hardware substrate is not as capable of dealing with unbalanced outputs as CGP, suggests that the presented approach might be particularly useful in cases where representations with fixed or predefined topologies are imposed. Fixed topologies lack certain abilities of neutral search, which are present in CGP. Thus, balancing the output could be both a means for assessing a hardware substrate's evolvability and a tool to improve the neutral search capabilities of a given substrate.

VI. CONCLUSION

of This paper has revisited the investigation on IPOP. Once more, it has been shown that the shape and design of the input

TABLE III

RESULTS WITH THE UNBALANCED AND BALANCED OUTPUT PATTERNS FOR 4-BIT PARITY, 4-BIT AND, 2-BIT FULL ADDER, AND 2-BIT MULTIPLIER. THE TERM UNBALANCED OUTPUT PATTERN REFERS TO USING EACH ENTRY OF THE TRUTH TABLE INPUTS ONCE WHEN EVALUATING THE CIRCUITS DURING EVOLUTION. THE LAST RESULTS FOR THE 2-BIT MULTIPLIER PRESENTED IN THE TABLE ARE THE RESULTS OBTAINED FROM SOFTWARE RUNS USING CGP WITH BALANCED AND UNBALANCED OUTPUT PATTERNS. IN THE CASE OF THE INTRINSIC EXPERIMENTS, IT HAS BEEN OBSERVED THAT USING OUTPUT BALANCING INPUT PATTERNS RESULTS IN A REAL TIME SPEED UP OF 20-25%, IN ADDITION TO THE INCREASE IN THE SUCCESS RATES.

Problem	Input Pattern	No. Successful Runs	Avg. Gens for Successful Runs	Avg. Fitness for Unsuccessful Runs
4-bit parity	balanced = unbalanced	20	162 ± 123	N/A
4-bit AND	unbalanced	20	23 ± 23	N/A
4-bit AND	balanced	20	12 ± 8	N/A
2-bit adder	unbalanced	2	1978 ± 968	13.33 ± 7.77
2-bit adder	balanced	8	5156 ± 2086	9.50 ± 5.02
2-bit multiplier	unbalanced	3	3112 ± 564	29.35 ± 33.66
2-bit multiplier	balanced	10	5420 ± 2999	7.50 ± 6.26
2-bit multiplier - CGP	balanced	100	3776 ± 5257	N/A
2-bit multiplier - CGP	unbalanced	100	4792 ± 7157	N/A

pattern order is important, particularly in the case of intrinsic evolution. Although most of the problems are not an issue in simulation where the representation can easily be changed, they make a real difference in hardware experiments.

We have proposed a method with which the input pattern order is designed in such a way that the output value distribution of the target circuit is balanced. The method shown has significantly improved the success rate of the intrinsic evolution of 2-bit fulladders and 2-bit multipliers. In the case of fulladders the success rate is four times higher, and in the case of multipliers the success rate is three times higher when balanced output method is used. In addition, in the case of the intrinsic experiments, it has been observed that using balancing input patterns results in a real time speed up of 20-25% for multiplier, adder and the AND circuit problems. The fact that the EA achieved 100% success rate in the case of 4-bit AND suggests that the decreased success rates in the case of the full adder and multiplier is due to the increased complexity caused by the higher output count. However, another factor which is likely to affect the performance of multiple output circuit evolution in hardware, which has not yet been investigated, is the expected physical location of the inputs and outputs in hardware. In [18], it is shown that evolution benefits from versatile input configurations, i.e. the inputs are available at multiple IO ports of the evolvable hardware platform. This might also be the case for the output locations. Thus, investigating the effects of using versatile output configurations during evolution rather than forcefully expecting the outputs from a single location on a hardware substrate should be interesting. Furthermore, the comparison of results from intrinsic and extrinsic experiments suggests that balancing the output value distribution could be both a means for assessing a hardware substrate's evolvability and a tool to improve the evolutionary search capabilities on a given substrate.

Concluding, the results presented confirm the importance of the input pattern order, particularly in the case of intrinsic hardware evolution. Together with previously published results [18], it is found that: first, randomising the input pattern order is crucial to achieve solutions that work under unpredictable conditions, rather than merely working intermittently. Second, randomising the input pattern helps to keep evolution away from local optima. Third, shaping the input pattern in a way that the frequency with which the outputs occur is balanced for all values, considerably increases the success rate and the performance of intrinsic evolution runs. Interestingly, it was discovered that the most popular EHW problem, the n-bit parity problem, is not a highly suitable test problem for intrinsic evolution experiments as it does not test the system's capability to handle unbalanced outputs (which is the case in most real world problems).

REFERENCES

[1] L. Barbulescu, J.-P. Watson, and D. L. Whitley, "Dynamic representations and escaping local optima: Improving genetic algorithms and local search," in *In Proceedings of AAAI'2000*, 2000, pp. 879–884.

[2] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. New York, NY, USA: John Wiley & Sons, Inc., 2001.

[3] T. G. W. Gordon, "Exploiting development to enhance the scalability of hardware evolution," Ph.D. dissertation, University College London, July 2005.

[4] A. Greensted and A. Tyrrell, "Extrinsic evolvable hardware on the RISA architecture," in *Proceedings of 2007 International Conference on Evolvable Systems*, September 2007.

[5] ——, "RISA: A hardware platform for evolutionary design," in *Proceedings of 2007 IEEE Workshop on Evolvable and Adaptive Hardware*, April 2007.

[6] G. Hollingworth, S. Smith, and A. Tyrrell, "The intrinsic evolution of virtex devices through internet reconfigurable logic," in *3rd International Conference on Evolvable Systems: from Biology to Hardware*. Edinburgh,: Springer-Verlag, April 2000, pp. 72–79.

[7] K. Imamura, J. A. Foster, and A. W. Krings, "The test vector problem and limitations to evolving digital circuits," in *EH '00: Proceedings of the 2nd NASA/DoD workshop on Evolvable Hardware*. Washington, DC, USA: IEEE Computer Society, 2000, p. 75.

[8] T. Kalganova, "Bidirectional incremental evolution in extrinsic evolvable hardware," in *The Second NASA/DoD workshop on Evolvable Hardware*, J. Lohn, A. Stoica, and D. Keymeulen, Eds. Palo Alto, California: IEEE Computer Society, 13-15 Jul. 2000, pp. 65–74.

[9] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[10] ——, *Genetic programming II: automatic discovery of reusable programs*. Cambridge, MA, USA: MIT Press, 1994.

[11] T. Kuyucu, M. Trefzer, A. Greensted, J. Miller, and A. Tyrrell, "Fitness functions for the unconstrained evolution of digital circuits," in *9th IEEE Congress on Evolutionary Computation (CEC08)*, Hong Kong, June 2008, pp. 2589–2596.

[12] J. F. Miller and P. Thomson, "Cartesian genetic programming," in *Genetic Programming, Proceedings of EuroGP'2000*. Springer-Verlag, 2000, pp. 121–132.

[13] E. Stomeo, C. Lambert, N. Lipnitsakya, and Y. Yatskevich, "On evolution of relatively large combinational logic circuits," in *EH '05: Proceedings of the 2005 NASA/DoD Conference on Evolvable Hardware*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 59–66.

[14] E. Stomeo and T. Kalganova, "Improving ehw performance introducing a new decomposition strategy," in *Proceedings of the 2004 IEEE Conference on Cybernetics and Intelligent Systems*, Singapore, December 2004.

[15] E. Stomeo, T. Kalganova, and C. Lambert, "Generalized disjunction decomposition for the evolution of programmable logic array structures," in *AHS '06: Proceedings of the first NASA/ESA conference on Adaptive Hardware and Systems*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 179–185.

[16] A. Thompson, "Evolving electronic robot controllers that exploit hardware resources," in *Advances in Artificial Life: Proc. 3rd Eur. Conf. on Artificial Life (ECAL95)*, ser. LNAI, F. Morán, A. Moreno, J. J. Merelo, and P. Chacon, Eds., vol. 929. Springer-Verlag, 1995, pp. 640–656.

[17] J. Torresen, "Evolving multiplier circuits by training set and training vector partitioning," in *Proc. ICES'03:From biology to hardware*, vol. 2606. Springer-Verlag, 2003, pp. 228–237.

[18] M. Trefzer, T. Kuyucu, A. Greensted, J. F. Miller, and A. M. Tyrrell, "The input pattern order problem: Evolution of combinatorial and sequential circuits in hardware," in *The 8th International Conference on Evolvable Systems: From Biology to Hardware*, 2008, to be published.

[19] V. K. Vassilev and J. F. Miller, "Scalability problems of digital circuit evolution: Evolvability and efficient designs," in *EH '00: Proceedings of the 2nd NASA/DoD workshop on Evolvable Hardware*. Washington, DC, USA: IEEE Computer Society, 2000, p. 55.

[20] J. A. Walker and J. F. Miller, "Predicting prime numbers using cartesian genetic programming," in *Proceedings of 10th European Conference on Genetic Programming*, vol. 4445/2007. LNCS, 2007, pp. 205–216.

[21] J. Walker and J. Miller, "Embedded cartesian genetic programming and the lawnmower and hierarchical-if-and-only-if problems," in *Proceedings of the 2006 Genetic and Evolutionary Computation Conference*. Seattle, Washington: GECCO, 2006, pp. 911–918.

[22] H. Xie, M. Zhang, and P. Andreae, "Automatic selection pressure control in genetic programming," in *6th International Conference on Intelligent System Design and Applications*, B. Yang and Y. Chen, Eds. Jinan Nanjiao Hotel, Jinan, China: IEEE, Oct. 16-18 2006, pp. 435–440.