Chapter 1
Modular Cartesian Genetic Programming

1 Embedded Cartesian Genetic Programming (ECGP)

1.1 Cone-based and Age-based Module Creation

In [3], Koza described the automatic definition and reuse of functions that are frequently appearing patterns in a chromosome. Turning these patterns into regular building blocks (modules) for evolution automatically decomposes the problem and allows for hierarchical problem solving. Since Koza relied on trees to represent the chromosome, his implementation of module creation focused on subtrees which are convex structures corresponding to subfunctions. In contrast, the CGP and ECGP representation models essentially describe a directed acyclic graph (DAG). A subgraph of a DAG is, however, not necessarily convex. The original ECGP model selects a node randomly and then adds nodes with contiguous node numbers to form a module. Such modules do not represent typical hardware building blocks; the nodes within these modules might even be completely unconnected. In this section we discuss two alternative techniques, cone-based and age-based module creation. While cone-based module creation focuses on convex subgraphs that actually represent typical hardware building blocks, age-based module creation extends the original random selection of nodes with the concept of aging.

1.1.1 Cone-based Module Creation

Figure 1 presents the standard ECGP module creation technique that aggregates nodes with contiguous node numbers to modules. Cone-based module creation selects nodes that form cones and thus might better correspond to subfunctions, especially in the area of digital circuits. Cones are a widely-used concept in circuit synthesis, especially in lookup-table mapping for FPGAs (see, for example, [4]). Given a node f_r in the DAG, a cone rooted at f_r consists of f_r itself plus some predecessor nodes such that for any node f_i in the cone there exists a path from f_i to f_r

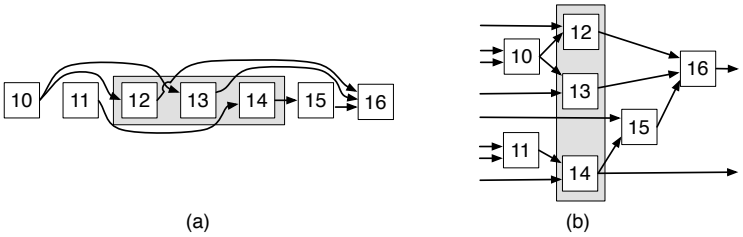

Fig. 1: Original ECGP module creation (a) can lead to subfunctions which are atypical in digital design (b)

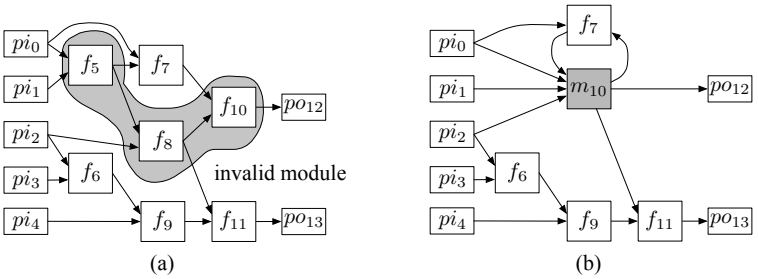

Fig. 2: Cones with reconvergent paths are invalid (a) as they can lead to combinational feedback loops (b)

that is entirely in the cone. Thus, a cone is a convex graph. For example, in Figure 2(a) the node set (f_{11}, f_9, f_8) forms a cone. Note that while a cone has a distinct root node f_r, it can have several outputs. The rationale behind cone-based module creation is that many useful substructures in classically engineered circuits are cones, e.g., the sum and carry functions of a full adder.

To generate module candidates, we randomly select a primitive node f_i and create a cone rooted at f_i with a number of nodes randomly chosen between n_{min} and n_{max}. One subtlety in generating cones is that we have to avoid what is called *reconvergent paths* in logic synthesis. To discuss several subtypes of cones consider again the circuit in Figure 2(a). The node set (f_{11}, f_9, f_6) is called a fan-out free cone because the fanout (output connections) of every node except the root node stay within the cone. Such cones are certainly safe candidates for module creation. The node set (f_{11}, f_9, f_8, f_5) does not form a fan-out free cone, as the outputs of f_5 and f_8 leave the cone. Nevertheless, this node set forms a valid module. In contrast, the node set (f_{10}, f_8, f_5) which is highlighted in Figure 2(a) does not form a valid module as the output of f_5 leaves and reenters the cone. If this cone was turned into a module the resulting circuit, shown in Figure 2(b), would contain a combinational feedback loop. The path formed by nodes (f_5, f_7, f_{10}) is called reconvergent with respect to the cone (f_{10}, f_8, f_5).

Reconvergent paths are specific to the cone-based module creation technique. Neither the age-based technique (see Section 1.1.2) nor the original ECGP method can create such paths. Compared to original ECGP module creation, which relies on contiguous node numbers, we use breadth first search starting with the cone's root node to avoid reconvergent paths. Resuming the example of Figure 2(a), a cone of size 3, rooted at node f_{10} would be formed by the nodes (f_{10}, f_7, f_8). Module creation stops when the randomly chosen number of nodes has been aggregated or when a module is hit. There are no modules within modules. As the original ECGP technique, our cone-based module creation technique ensures that inputs to a new module only come from nodes with smaller node numbers, and outputs of the new module only connect to nodes with higher node numbers. After selecting nodes for a new module the nodes are compacted and renumbered to form a coherent block which then can be compressed to a module by the standard ECGP compress operator.

1.1.2 Age-based Module Creation

Age-based module creation aggregates primitives nodes that have persisted in the chromosome for a higher number of generations. The rationale behind age-based module creation is that aged nodes are likely to contribute directly or indirectly to an individual's success and should therefore be preferred over randomly selected nodes.

We assign to each primitive node f_i an attribute $age(f_i)$. The age is incremented by one in each generation and set to zero when the node is selected for mutation or compression. The age of primitive nodes within modules remains unchanged; modules themselves do not have an age. We form module candidates by aggregating primitive nodes, restricting the number of nodes by lower (n_{min}) and upper bounds (n_{max}). The average age of a module candidate m_j is then given by

$$\text{age}(m_j) = \frac{\sum_{f_i \in m_j} age(f_i)}{|m_j|}$$

Our age-based module creation technique relies on a two-stage binary tournament to select a module. That is, we generate a module candidate by following procedure: First, we select a random primitive node f_i and a number of primitive nodes $n, n_{min} \leq n \leq n_{max}$, randomly. Then, we extend the module from f_i to nodes with smaller node numbers until we hit a module or aggregate exactly n primitive nodes. We create another module using a different random primitive node f_i and draw the one with higher average age. If both modules have the same average age, we draw one module randomly. This step is repeated once to derive the final module.

We have also experimented with selecting the module candidate with maximum average age. This requires the formation and evaluation of a larger number of module candidates. However, picking the module with maximum average age has proven inferior to the two-stage binary tournament scheme for all test problems. An expla-

nation for this lies in the fact that maximizing average module age tends to generate modules with a very small number of high-aged nodes. It seems that while using node age as a guide to steer module creation is highly effective (see Section 1.1.3), the technique is rather sensitive to the size of modules.

1.1.3 Benchmarks, Metrics and Results

	benchmarks		
	parity	multiplier	EMG classifier
chromosome length	50 nodes	200 nodes	200 nodes
number of inputs n_i	3/4/5	4/6	200
number of outputs n_o	1	4/6	1
functional set	2-LUT: and nand, or, nor	4-LUT: and and$_{inv}$, or, xor	4-LUT any function
mutation rate	0.03	0.03	0.03
one-point mutation probability	0.6	0.6	0.6
compress/expand probability	0.1/0.2	0.1/0.2	0.1/0.2
module mutation probability	0.1	0.1	0.1
module size (n_{min}, n_{max})	2...8 nodes	2...10 nodes	2...10 nodes

Table 1: ECGP parameters for the parity, multiplier and classifier benchmarks

We compare the original, age-based and cone-based ECGP module creation techniques on even-parity and multiplier benchmarks. Additionally, we include classifiers for electromyographic (EMG) signals as test problems. In this classification application, skin-attached sensors collect electric signals of contracting muscles to control a prosthetic hand [5]. The test data has been recorded from four muscles of a volunteer's forearm. A sequence of eight contractions (movements) with 20 repetitions each has been measured. The typical signal for a movement is composed of a 9 seconds relax phase and a 5 seconds contraction phase. For two seconds of the contraction phase we have removed the dc offset and applied RMS smoothing to achieve the feature vectors. The resulting data set consists of 144 strings of 200 bits each. Based on that data we try to evolve a classifier circuit for the movement "open hand".

Table 1 shows the ECGP model parameters, including the chromosome length, the number of inputs and outputs, and the function set for the nodes. For the parity function, we use 2-input lookup tables (LUT) to model the nodes, but restrict the function set to a few Boolean functions. For the multipliers, we use 4-input LUTs as node models but again restrict the function set to the functions and, or, xor, as well as and$_{inv}$, which is an and with one input inverted. Finally, for the EMG classifiers we use 4-LUTs without any restriction on the node function. As an optimization algorithm we employ a $1+4$ evolutionary strategy (ES). The corresponding parameters as well as the parameters for module creation are also shown in Table 1. We con-

duct all experiments with our modular framework for evolutionary hardware design presented in [6].

As evaluation metrics we choose the *computational effort* (CE) as presented by Koza in [3]. The CE metrics states the expected value for the number of fitness evaluations required to reach the optimization goal with a probability of z. To determine the CE metrics, we repeat all experiments for 50 times and set z to 99%. The CE metrics cannot be applied directly to the classifier benchmark. Classifiers differ from arithmetic circuits in that there is no simple correctness measure. Typically, classifiers are evolved with training data and then run on test data to determine metrics such as classification rate. As we want to investigate and compare the computational effort for evolving a classifier and not the generalization capabilities of the ECGP model, we measure the classifiers' fitness on the training data set and define it to be correct when the classification rate on training data exceeds 85% and 95%, respectively.

	computational effort (CE)				
	random	age-based		cone-based	
	absolute	absolute	relative	absolute	relative
2x2 mul	66,623	51,961	**−22.0%**	49,052	**−26.4%**
3x3 mul	8,840,574	6,001,917	**−32.1%**	3,638,120	**−58.9%**
3-parity	81,122	49,160	**−39.4%**	87,915	+8.4%
4-parity	477,880	494,295	+3.4%	265,796	**−44.4%**
5-parity	1,825,645	1,385,244	**−24.1%**	1,112,691	**−39.1%**
85% EMG classifier	18,260	14,743	**−19,3%**	23,855	+30.7%
95% EMG classifier	510,147	314,311	**−38,4%**	873,319	+71.2%

Table 2: Experimental results for an $1+4$ ES with random, age-based and cone-based module creation

Table 2 presents the comparison of the different module creation techniques. The table reports the computational effort in absolute numbers and relative to the original random module creation technique. A negative relative effort denotes an improvement. From the experimental results, we can make the following observations: *i)* Age-based module creation is highly effective. For six out of the seven test problems, age-based module creation lowers the computational effort in comparison to the previous method with improvements ranging between some 20% and 40%. The one exception is the 4-parity function, where the computational effort increases slightly by 3.4%. *ii)* The overall results for the cone-based module creation technique are somewhat inconclusive. However, looking at the different test problems we note that for the evolution of multipliers and for larger parity functions cone-based module creation proves highly beneficial. In contrast, for evolving EMG classifiers the cone-based approach does not work at all. Intuitively, the identification of cones as useful subcircuits is hampered if the function is rather small or is a single-output function. In the first case there is no sufficient potential for creating cones, whereas the second case lacks re-usability of a cone for different

outputs. Multipliers are highly regularly structured functions that are neither particularly small nor single-output functions. From the experimental data it is clear that cone-based module creation is effective for multipliers, especially more effective than age-based module creation. In contrast, EMG classifier circuits are random logic functions which might explain the unsatisfying performance of cone-based module creation for this class of problems.

1.2 Cone-based Crossover

Based on the cone-based module creation technique, we investigate the efficiency of a cone-bases crossover operator for the ECGP hardware representation model. The crossover operator selects a convex sub-DAG in one parent and replaces it with a convex sub-DAG extracted from another parent. Crossover implies the use of a genetic algorithm (GA) instead of the evolutionary strategy typically applied in ECGP. GAs are of interest for two reasons. First, a GA exchanges partial solutions within the population which can help escape local optima. Specifically, a GA scheme with an increased population might reveal a more stable convergence behavior compared to an ES. Second, one might be interested in evolving circuits that are not only functionally correct but also fast and small. Multi-objective evolutionary algorithms (MOEAs) are an intriguing approach for such optimization problems. Modern Pareto-based MOEAs evolve a set of diverse individuals in a single run and rely on an operator that exchanges partial solutions of individuals.

Our cone-based crossover operator works as follows: In the first step, we form a cone in a donor chromosome by randomly selecting a root node and a size between n_{min} and n_{max}. This procedure is similar to the cone-based module creation of Section 1.1.1 except that we treat both primitive nodes and modules as atomic nodes. Note that at this point, a cone can contain modules. In the second step, we randomly select a root node in a recipient chromosome and try to form a cone of exactly the same size as the donor's cone. Depending on the actual DAGs, the resulting cone of the recipient can be smaller than the donor's cone. The third step comprises the formation of two sets, set p that contains nodes of the donor's cone which have output connections to nodes outside the cone, and another set q that contains the nodes of the recipient which connect to nodes within the recipient's cone. Figure 3 displays an example. The donor's cone consists of three nodes. As all these nodes provide cone outputs, we derive $p = \{f_{10}, f_{11}, f_{22}\}$. The recipient's cone receives inputs from three nodes, and we derive $q = \{f_5, f_7, f_8\}$. The fourth step actually transplants the donor's cone into a clone of the recipient, forming a new recombined chromosome. This process preserves all node types. Specifically, nodes in the donor's cone which are modules of type I or II (see Section **??**) remain modules of type I or II, respectively. The module descriptions of the recombined chromosome are updated accordingly. In the final step, dangling inputs of the transplanted module and the recipient chromosome are randomly connected to the nodes

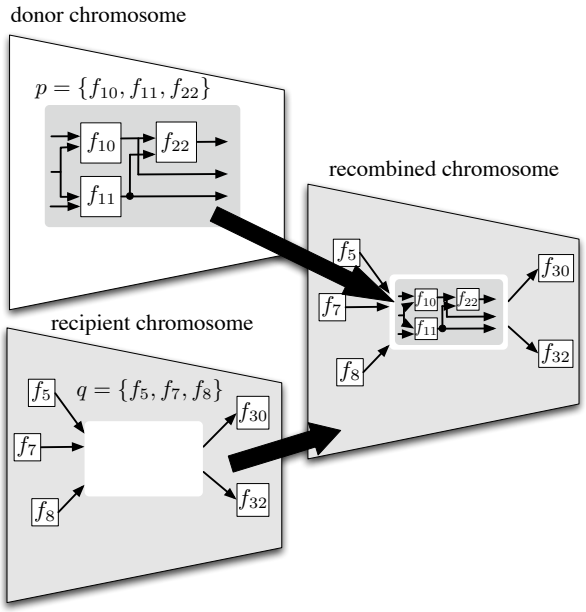

Fig. 3: Cone-based crossover: A cone of a donor chromosome is transplanted into a clone of a recipient chromosome.

in lists p and q, respectively. If the resulting chromosome still contains unconnected inputs, they are connected randomly to preceding nodes.

	computational effort				
	$1+4$ ES	GA, $\|$population$\|$=5		GA, $\|$population$\|$=50	
	absolute	absolute	relative	absolute	relative
2x2 mul	66,623	64,111	$-\mathbf{3.8}\%$	102,593	$+54.0\%$
3x3 mul	8,840,574	2,518,964	$-\mathbf{71.5}\%$	39,064,742	$+341.9\%$
3-parity	81,122	382,036	$+470.9\%$	186,898	$+130.4\%$
4-parity	477,880	6,294,678	$+1217.2\%$	6,482,504	$+1256.5\%$
85% EMG classifier	18,260	19,859	$+8,8\%$	28,825	$+57,9\%$
95% EMG classifier	510,147	576,988	$+13.1\%$	695,794	$+36.4\%$

Table 3: Experimental results: $1+4$ ES versus GA with different population sizes

For experimentally evaluating the cone-based crossover scheme we use the same setup as in Section 1.1.3 and compare the 1+4 ES scheme with a standard, elitism-based GA configured with a small (GA-5) and a large population (GA-50). The elitism rate for the GA is 5.0%, with at minimum one selected individual. In GA-5, the best individual proceeds directly to the next generation. In GA-50, the three

best individuals proceed directly to the next generation which leaves us with 47 remaining fitness evaluations. The cone-based crossover operator considers cones with a size of up to 20 nodes (primitive nodes and modules) and is applied with a probability of 1.0%.

Table 3 presents the experimental results. The table reports the computational effort in absolute numbers and relative to the original ECGP method. A negative relative effort denotes an improvement. From the experimental results, we can make the following observations: *i)* Comparing the $1+4$ ES to a GA with population size of 5, we conclude that the GA is better for multipliers and dramatically worse for the parity function and for the EMG classifiers. This points to the effectiveness of the cone-based approach for multipliers and to its inefficiency for single-output and random logic circuits. *ii)* Increasing the population size for the GA to 50 increases the computational effort in any case substantially. It has to be noted that a GA with a population size of 50 also evolves correct circuits but needs far more fitness evaluations. In each generation, GA-50 performs $11.75 \times (47/4)$ more fitness evaluations than ES and GA-5. As the results show, even for multipliers this larger potential for recombination does not outweigh the higher effort per generation.

References

1. J. A. Walker and J. F. Miller, "Evolution and Acquisition of Modules in Cartesian Genetic Programming," in *Proceedings 7th European Conference on Genetic Programming (EuroGP)*, vol. 3003 of *LNCS*, pp. 187–197, Springer, April 2004.
2. P. Kaufmann and M. Platzner, "Advanced Techniques for the Creation and Propagation of Modules in Cartesian Genetic Programming," in *Proceedings 10th Conference on Genetic and Evolutionary Eomputation (GECCO'08)*, pp. 1219 – 1226, ACM Press, 2008.
3. J. R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, 1994.
4. J. Cong and Y. Ding, "Combinational Logic Synthesis for LUT Based Field Programmable Gate Arrays," *ACM Transactions in Design Automation of Electronic Systems*, vol. 1, no. 2, pp. 145–204, 1996.
5. K. Glette, T. Gruber, P. Kaufmann, J. Torresen, B. Sick, and M. Platzner, "Comparing Evolvable Hardware to Conventional Classifiers for Electromyographic Prosthetic Hand Control," in *Proceedings 3nd NASA/ESA Conference on Adaptive Hardware and Systems (AHS'08)*, pp. 32–39, IEEE Computer Society, 2008.
6. P. Kaufmann and M. Platzner, "MOVES: A Modular Framework for Hardware Evolution," in *Adaptive Hardware and Systems (AHS'07)*, pp. 447–454, IEEE Press, 2007.