Supergenes in a Genetic Algorithm for Heterogeneous FPGA Placement

Peter Jamieson
Department of Electrical and Computer Engineering
Miami University
Email: jamiespa@MiamiOH.edu

Farnaz Gharibian and Lesley Shannon
School of Engineering Science
Simon Fraser University
Email: fga7 or lshannon@ensc.sfu.ca

***Abstract*—Supergenes are an addition to a genetic algorithm's genome that duplicate genes in the genome, represent local optimizations, and have the potential to be expressed overriding the duplicated gene. We introduce supergenes in a genetic algorithm for FPGA placement where a placement algorithm places a mix of fine-grain components and medium-grain components (where a medium-grain component is 2 to 10 times the size of a fine-grain component). This is the first placement algorithm, to our knowledge, that can deal with such a mix of components on an FPGA. Our results show that supergenes improve a placement metric (clock speed of the FPGA) by approximately 10%. We also show and explore mutation operators on supergenes, and we experimentally demonstrate that the expression of a supergene can be effectively controlled via a binary function for our placement problem.**

***Keywords*—Genetic Algorithms, Supergene, FPGA, Placement, Granularity**

I. Introduction

Field-Programmable Gate Arrays (FPGAs) are reprogrammable integrated chips (ICs) that can implement a range of digital circuits. In the process of mapping a digital circuit to an FPGA, an algorithm, called placement, attempts to put the elements of a design circuit onto an FPGA while optimizing a cost function that captures speed (the rate at which the FPGA can be clocked), area efficiency, and often, power consumption. This can be achieved with a number of combinatorial optimization algorithms including simulated annealing (SA), analytic placers, and evolutionary algorithms such as genetic algorithms (GAs). The complexity of the placement problem has resulted in some stagnation of improvements in both the quality of placement and, most importantly, run-time [1], and current ideas have focused on speeding up algorithmic run-time via larger clusters of circuit elements and parallelization. GAs are well suited for parallelization, and in this work, we demonstrate how they can be used for placing small collections of clusters with fine-grain elements.

Therefore, this work makes contributions to both the study of GAs and FPGA placement, and additionally, provides and exposes GA researchers to one of the target applications of their work. For FPGA placement, we introduce the first implementation of a placement algorithm that can place both fine-grain individual circuit elements and medium-grain clusterings of these elements (2 to 10 times the size of the fine-grain) onto an FPGA. To achieve this and in relation to GA research, this work introduces a medium-grain gene into the genome that we, currently, call supergene taken from the biological term [2]. Supergenes are an additional component of GA's genome that are used to duplicate grouping of genes in the genome, represent this grouping as a local optimization, and are optionally expressed, which will override the traditional genes in the genome. Since a circuit element can exist in both the fine-grain genome and the set of supergenes, a dominance factor is used to determine which of the conflicting versions will express itself. In this way, we have created a GA placement algorithm for fine and medium-grained circuit elements, and we have introduced a concept of supergenes that may benefit other combinatorial optimization problems with the existence of similar local optimizations.

We should note that GAs for FPGA placement still lack a crossover operator that would make these algorithms comparable to other placement algorithms such as SA. Still, as the search for this crossover operator continues, GA improvements such as supergenes are still of value to investigate as they should provide additional performance and quality benefits. Therefore, for our experiments we use both the partial mapped crossover (PMX) [3] and the confined-swap recombination (CSR) [4] operators as these are the best demonstrated crossovers operators for FPGA placement, at present.

Our results show that a GA algorithm with both the CSR crossover and supergenes improves the speed of the placed circuits on average by approximately 10% compared to a GA with the same crossover operator (without supergenes). Additionally, we find that the dominance factor included in the supergene is best set to always dominate and express the supergene when the elements defined as supergenes are of high-quality. This is the case since the medium-grained clusters that are included as inputs to the algorithm are definitive good structures, and therefore, they significantly improve the quality of the placement. In the case of a GA without supergenes, the algorithm spends unnecessary time trying to reconstruct the information pre-encoded in the supergene, and this takes a number of generations to accomplish when this information is missing. Supergenes allow an algorithm to explore other, better solutions. We also investigate how mutating the dominance factor in a supergene impacts the results by observing the same GAs that included, potentially, false positive super-clusters (not necessarily great supergene candidates). This experiment shows that a supergene GA keeps the majority of good super-clusters and eliminates the expression of the majority of false positives.

The remainder of this paper is organized as follows: Section II describes some FPGA terms and briefly reviews FPGA placement algorithms, including GAs for FPGA placement, and section III describes the what the granularity of placement is and how medium-grained circuit elements are obtained. Section IV describes the genome, supergene, and GA that is used for our experiments. Section V describes our experimental setup, section VI shows the results of these experiments, and section VII concludes this work.

II. Background

In this section, we describe FPGA terminology used in this paper, algorithms for FPGA placement, previous research on GAs for FPGA placement, and ideas related to the supergene for GAs.

A. FPGAs, Software, and FPGA Placement Algorithms

FPGAs are programmable ICs that consist of programmable logic blocks, called clusters, and programmable routing [5] where the programmable routing consists of wire segments that are connected to either logic blocks or other wire segments via programmable switches. The placement problem, for these devices, is to place the clusters that make up a digital circuit such that the critical path (the physically longest path in the design that determines the clock speed) is minimized, the power consumption of the programmable routing is minimized, and the area-efficiency is maximized.

VTR [6] is an open source Computer Aided Design (CAD) flow that allows researchers to experiment with both the FPGA architecture and the CAD flow algorithms that maps digital designs to an architecture including placement. VTR, currently, uses SA for its placement algorithm.

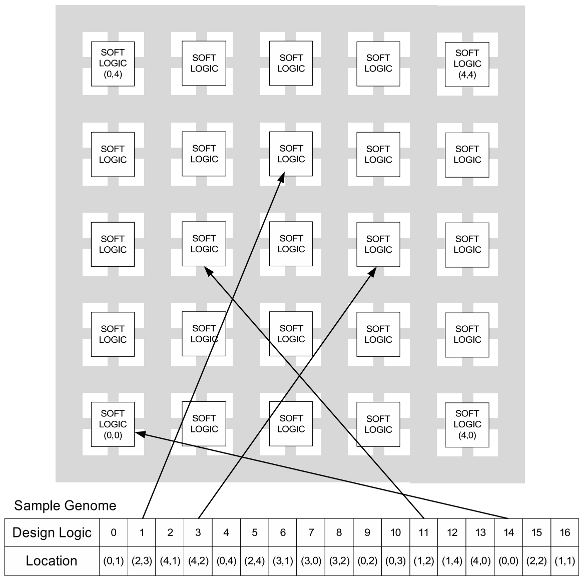
FPGA placement algorithms try to place hundreds to hundreds of thousands of clusters, representing the digital design, onto the array of FPGA clusters focusing on optimizing the critical path, the power consumption, and the area-efficiency. This problem has been shown to be NP-complete to solve optimally, and a number of popular algorithms have been proposed to solve this problem including SA ([7], [5]), mincut ([8], [9], [10]), and analytic ([11], [12]) placers. A good overview of the FPGA placement problem can be found in Betz *et. al.* book [5].

When attempting to improve a placement algorithm, the important metrics to focus on are power consumption, critical path, and area-efficiency. In this work, we do not consider power. Critical path, as defined earlier, is a measure of the longest clocked path in the circuit. This value determines the maximum clocking speed of the circuit and decreasing the length of this path is considered to improve the placement quality of the circuit.

Area-efficiency is a metric that may or may not be applicable to the placement algorithm. For FPGA researchers concerned with architecture and CAD, area-efficiency is, partially, manifested in channel width. Channel width, represented by the variable **W**, is the number of programmable wires that exist in the channels in between clusters. For each circuit, we can find the minimum **W** for an FPGA that will allow that circuit to be legally routed. We use **W** as a quality result since it reflects information on the area-efficiency of an algorithm in terms of minimizing the number of wires needed in an FPGA architecture.

B. GAs for FPGA Placement

Within the Very Large Scale IC (VLSI) domain, GAs have been applied to a number of CAD problems, and for a more thorough survey of these problems the reader can refer to Mazumder and Rudnick [13]. In this work, we are specifically dealing with GAs for FPGA placement.



Design Logic	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Location	(0,1)	(2,3)	(4,1)	(4,2)	(0,4)	(2,4)	(3,1)	(3,0)	(3,2)	(0,2)	(0,3)	(1,2)	(1,4)	(4,0)	(0,0)	(2,2)	(1,1)

Fig. 1. Sample genome for 17 elements on a 5x5 FPGA

Evolutionary programming algorithms have been implemented and explored for FPGA placement, and the first published attempts were by Venkatraman *et. al.* [14] in which they implemented a GA based placer in VPR 4.3 [15] (one of the predecessor to VTR). In their work, each cluster's location on the FPGA array is a gene, and the 2D location of each of the clusters forms an individuals genome. Figure 1 shows a fine-grain genome for a design consisting of 17 elements. A population of these individuals is created and each individual is evaluated based on a fitness function for speed and area-efficiency. Within the population, the fittest individuals are kept and mutated to create the next generation based on proportional representation, which is left undefined by the authors. Mutations are based on local cluster swaps and global swaps. Their results show that this algorithm improves the critical path compared to VPR's SA algorithm for ten benchmarks. Unfortunately, there is no analysis of run-times for the two algorithms (SA and GA), which means the algorithms can not be fairly compared.

Meng *et. al.* [16] created an algorithm that combines both GA and SA algorithms for placement. Their approach claims that the GA aspect of the algorithm are used to escape local minimums (as another form of hill climbing) and the SA is used to quickly improve solutions. The genome for their approach is the same as the one previously described. They also use a fine-grain mutation based on swapping clusters, and they propose a new method for crossbreeding fit individuals. Their results show similar costs compared to VPR 4.3's SA approach with similar run-times.

More, recently, our GA implementations [17] [18] and work by Collier *et. al.* [4] [19] has found that GAs for FPGA placement are not yet comparable to SA implementations mainly due to the weakness of the crossover operator. Collier has proposed a CSR crossover operator, which they experimentally show is better than the PMX originally proposed by Goldberg for the traveling salesman problem [20]. The CSR as a crossover operator for FPGA placement guarantees that the off-spring will be similar to at least one of the two parents.

C. Research Related to Supergenes

The idea of supergenes [2] in GAs is that the phenotype (traits of an individual) emerge based on the interaction between multiple encodings within the genome. In particular, in combinatorial optimization problems a supergene defines the medium-grained behavior of a collection of combinatorial elements.

Our definition of supergenes is used in the context of combinatorial optimization problems solved via GAs. This is a narrow context and the idea has similarities to messy GAs, the idea of gene expression, and canonical genes. Messy generic algorithms (MGAs) [21] were introduced in 1989 by Goldberg *et. al.* and introduce the idea of variation in both the information length and the rigidity of the genome, and this type of GA suggests the possibility of containing supergenes. Similarly, proportional GAs (arguably a subset of MGAs and similar to Canonical GAs [22]), as introduced by Wu *et. al.* [23], presents the idea of the phenotype of an individual as an expression of the existence or non-existence of genes within the genome. PGAs have genomes that allow for more complex encodings of individuals with less restrictions on strict formatting. Both MGAs and PGAs are ideas that modify the way a genome is created and impact the phenotype, and this relates to gene expression. Gene expression has been studied by researchers in the area of simulation of biological evolution. For example, Eggenberger [24] looks at gene expression for evolution of 3D organisms.

The similarities between supergenes for GAs and a number of other research directions in evolutionary programming is evident even with the small sampling we review. Our focus is to use the supergene as a means to advance GAs for combinatorial optimization algorithms. As we will describe further, the supergene has a specific use for the FPGA placement problem.

III. Fine, Medium, and Course Grain FPGA placement

FPGA placement, as described earlier, is the process of placing components of a digital circuit onto an FPGA to optimize a number of cost metrics. The reality, though, is that modern FPGAs no longer consist of just clusters, but can consist of a multitude of architectural features, and the CAD that targets these devices might perform placement at various granularity levels. In this section we define these granularity levels and some of the algorithms and research that has focused on these. The smallest component of an FPGA, from a structural perspective, are the clusters, which we will call fine-grain and are commonly referred to as the soft-logic.

Modern FPGAs consist of clusters and hard circuits such as hard multipliers, memories, configurable clusters, and processors [25], [26]. These hard circuits are included since the equivalent implementation within the clusters is significant, and the use of hard structures within digital designs is significant enough that the risk of the silicon not being used is outweighed by the benefits of there existence [27]. Hard circuits are, also, fine-grain elements of the FPGA and can be incorporated into placement algorithms such as SA and GA since they have distinct placement locations and this aspect of the problem can be dealt with in conjunction with the placement of clusters; for example, VTR can place hard circuits and clusters at the same time. This placement might be called heterogeneous placement since the fabric does not consist of only one type of structure, but it is not heterogeneous placement with respect to the size of the elements being placed.

Course-grain placement is on the opposite side of the spectrum of placers, and we define it as the placement of a large collection of homogeneous or heterogeneous elements. For example, a digital circuit might consist of two large separate parts called Intellectual Property (IP) blocks. IP blocks can be purchased from vendors and allow designers to quickly implement complex functionality, and an IP block might consist of upward of thousands of clusters and hard circuits. In addition to IP blocks, coarse-grain placement manifests itself via both design partitioning [28] and incremental placement [29]. In all cases, the coarse-grain placement algorithms deals with first placing large coarse-grain blocks, similar to IP blocks, and then performing local placement within the coarse-grain blocks (fine-grain placement). A common approach to coarse-grain placement that deals with the coarse-grain placement is with floorplanning algorithms [30] [31] including GA attempts at floorplanning [32].

This work focuses on what we call heterogeneous medium-grained and fine-grained placement. Medium-grained structures are small collections of clusters (between 2 and 10) that are known to be strongly related to one another and should be placed in close proximity to each other on the FPGA. Our terminology for these medium-grained structures is super-clusters [33], which relates to our choice of using the term supergenes. Medium-grained placement would be no different than fine-grain placement if we could completely partition the entire design into the same sized super-clusters that could be placed in grid-like manner, but our research has found that there is only a small percentage of fine-grain clusters that can be organized into super-clusters and these super-clusters can vary in size (how many clusters are included in each). Therefore, a placement algorithm for super-clusters mixed with cluster needs to deal with this mix of medium-grain and fine-grain structures. This is a significant challenge where both fine and medium-grain structures can occupy the same places on the FPGA and will impact one another in terms of optimizing the cost function. In the VLSI domain, this problem is sometimes referred to as, "boulder and dust" [**?**].

A. Finding Medium-Grained Super-clusters

One important question is how are super-clusters extracted from a digital circuit. Our preliminary work [33] describes clustering methods that can find these medium-grained structures during the CAD flow. However, our current findings suggest that these methods do not find high quality super-clusters. For this work, we use super-clusters that are extracted in a, as yet, not described process. Since this is not the main contribution of this work, we simply state that the super-clusters used in this work are high quality super-clusters that range in size from 2 to 10 clusters. We have confirmed that these super-clusters are of high-quality by running experiments with them that show that not placing the clusters in a super-cluster in close proximity to one another will cause significant increases in critical path delay and area-efficiency costs.

IV. Supergenes for Medium and Fine Grain Placement

In this section, we provide details of the GA for medium and fine-grain placement including the changes to the genome.

A. *GA for Placement*

Our GA for FPGA placement consists of a number of parameters and features that have been developed over a number of years. The GA has the following features:

- Inbreeding Avoidance: Our GA avoids crossbreeding similar individuals by maintaining a history of ancestors five generations back and not breeding with shared ancestors [34] [35]. For combinatorial optimization problems this is maintained via recording ancestors and is derived from ideas in Tabu search [36] and CHC [37].
- PMX or CRS: The crossbreeding operator is one of PMX [20] or CSR [4] and the ideas of these crossover operators are explained with graphs and algorithms in Collier *et. al.* [4]
- Probability of Mutation: Mutations are probabilistically accepted based on an improvement to the quality of the placement as taken SA FPGA placement [5].
- New Generations: Parents and children combined together in competition for new generations where ten percent of the parents are maintained [37].

Beyond these characteristics, the population size is 500 individuals per generation, 50 percent of the genes are randomly mutated, and 90 percent of the new generation of individuals are created from crossbreeding and mutation (meaning 10 percent of the population comes from the best of the previous generation). We will describe a number of additional parameters as related to supergenes.

B. *FPGA Genome Including Supergenes*

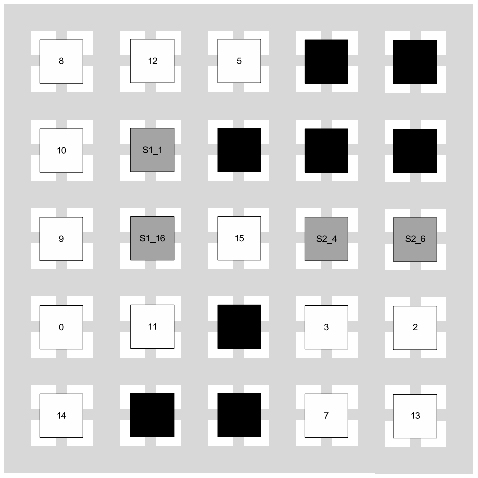
The FPGA genome for the fine-grain clusters is as previously shown in Figure 1. In this genome the location of each cluster is described as a location on the FPGA, and this allows for easy mapping to the fabric to evaluate the cost function for an individual's placement. Note that only clusters are included in the genome and we do not consider hard circuits, but to include these types of elements would be a minor algorithmic change.

To include the medium-grain clusters or super-clusters in the genome, we add supergenes. A supergene has the following properties:

- Size: The number of fine-grain genes contained in a supergene
- Internal Genes: A collection of genes in the supergene and properties of these genes
- Supergene details: Specific characteristics or details of the supergene
- Dominance factor: A function defining if the supergene will express itself over other genes

In the case of FPGA placement problem, the supergene has a size relative to the size of a super-cluster instance and the genes in the supergene are the clusters contained in the super-cluster with additional details of each genes relative x and y coordinates in relation to the x and y location of a supergene. For a specific individual, if a supergene expresses itself the locations of the clusters in the supergene will take precedence over the information contained in the fine-grain genome. Any genomes in the fine-grain genome that would map to the locations now occupied by the supergene are remapped in the same method as in the PMX crossover.

Figure 2 shows an example of a genome (extended from Figure 1) with two supergenes that have expressed themselves



Sample Genome

Design Logic	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Location	(0,1)	(2,3)	(4,1)	(4,2)	(0,4)	(2,4)	(3,1)	(3,0)	(3,2)	(0,2)	(0,3)	(1,2)	(1,4)	(4,0)	(0,0)	(2,2)	(1,1)

Supergene	s1	16	1	s2	4	6
Location	(1,2)	(0,0)	(0,1)	(3,2)	(0,0)	(1,0)

Fig. 2. A genome that includes expressed supergenes and appropriate remapping.

(note that the bottom left corner corresponds to the coordinates 0, 0 and the upper right is 4, 4). The placement points are colored white for fine-grain clusters, grey for super-clusters, and black if the location is not used. In particular, note how clusters 11 and 8 are remapped to a different location because of the supergene precedence. To achieve this remapping the PMX approach is used where a conflict is detected and the already placed supergene cluster's coordinates are used for the remapping. For example, in Figure 2 cluster 16 is already at coordinates (1,2), which cluster 11 is currently assigned in the fine-grain genome. We remap 11 to coordinates (1,1) as this is the coordinates of 16 in the fine-grain genome. Multiple conflicts will iterate through this remapping procedure guaranteeing that each cluster will have a unique location.

C. *Operators and Research Aspects of Supergenes*

There are four aspects of supergenes that can be experimented with and in the paper we treat them as follows:

- Gene Expression: There are different methods for a supergene to express itself. We use a binary dominance factor.
- Supergene Mutation: What and how to mutate the supergene including the dominance factor, the global relative information of the supergene, and the internal details of the supergene. We experimentally try mutations on each of these factors.
- Supergene Crossbreeding: What and how to crossbreed supergenes in different individuals. We do not study this in this paper.
- Expression Conflict Remapping: How to deal with conflicts between expressed supergenes and the genome to remap the collisions. We use a PMX remapping technique.

We only mutate supergenes, and when a new individual is created from crossbreeding of two fine-grain parents, we randomly pick one of the parents to pass on their supergenes to the new individual. There are a number of imaginable crossover operators to use instead of this approach, but we leave this as future research since the idea of the crossover operator remains the crux of GAs for FPGA placement.

Mutating the supergene encompasses a number of options as described above. In particular, we can mutate the global supergene location (called a global mutation), the relative gene locations in the supergene (called a local mutation), and the dominance factor. Our dominance factor is a binary decision that is set to either on or off. Experimentally, we have found that mutating the global and relative positions produces faster improvements in results, but the dominance factor is best set to on. Because the super-clusters in the experiments are almost guaranteed to benefit the final placement when placed in proximity, their is no benefit to mutating this self-expression off unless there is a higher chance that the supergene will not benefit the final placement. We explore this idea further in the results section.

V. Experimental Setup

For the experiments and results that we will present in the next section, there are a number of items we describe as related to the methodology for FPGA CAD experiments. We describe these details so that FPGA focused researchers can understand the environment under which the experiments were run, and GA focused researchers understand why the experimental methodology might differ from their expected methodology. In either case, the goal of the experiments are to demonstrate that supergenes can be used to implement an algorithm for medium and fine-grain FPGA placement, this implementation is well suited for GAs, and the supergene concept improves on the current state-of-the-art GAs for FPGA placement. We remind the reader that even though this approach improves the algorithm, the GA for FPGA placement is significantly worse than, for example, SA for FPGA placement. This disparity is, likely, due to the lack of a good crossover operator in this domain.

For this experimental setup the details that we describe are:

1) The software framework that the algorithm is implemented in and the benchmarks circuits used
2) The architectural parameters describing the FPGA we are placing the benchmarks on
3) The computation system and conditions under which the algorithms are executed

A. Software Framework and Benchmarks for FPGA CAD

As described earlier, VTR is an open-source academic tool that allows researchers to experiment with both FPGA architecture and CAD [6]. For these experiments we are using an internal version of VPR 5.0 that includes power estimation. In the experiments the power estimation and the power component of the cost functions (during various points of in the CAD flow) is turned off. This is done because the super-clusters, which are included in the supergene GA, were obtained in a non-power based flow. The remainder of the tool includes addition of GAs, which is optionally turned on to replace the SA algorithm for placement.

Our experiments are run using 10 benchmarks where these benchmarks have been converted to a netlist of clusters using an academic CAD flow. Table I shows a summary of the details of these benchmarks. Column 1 lists the benchmark name. Columns 2, 3 and 4 show the grid size of the FPGA, the number of clusters, and the number of I/Os. Column 5 shows the number of super-clusters in the benchmark where the maximum number of clusters in a super-cluster is 3.

TABLE I. Details for the benchmarks including super-clusters

Benchmark	Grid Size	# of Clusters	# of IOs	# of Super-clusters
cfc8	11x11	110	51	5
dconvert	22x22	396	258	39
desa	16x16	229	190	1
dsystemC	20x20	393	162	22
iir1	11x11	118	58	13
pajf	10x10	88	103	1
rsd1	14x14	171	20	5
rsd2	20x20	370	32	8
sv3	7x7	46	40	4
synth_14	46x46	521	544	11

Each benchmark is passed into VPR 5.0 for the same FPGA architecture as described below. VPR 5.0 uses one of the placement algorithms and then routes the design. The output of VPR 5.0 is the size of the FPGA (the same for each benchmark regardless of placement algorithm), the speed of the circuit in terms of the time delay for the critical path, and the channel width needed to route the circuit increased by 20% (as a relaxation on the circuit modeling real-world device usage).

B. Architectural and CAD parameters

The FPGA architecture and CAD parameters that describe both the FPGA and the CAD flow are shown in Table II. For a more detailed explanation of these parameters please consult [5], but for the sake of space and unnecessary details, we do not describe these parameters in detail here. Of particular importance to these experiments is the fixed I/O (input and outputs), which means that for every execution the pins that provide input and output values are always in the same location on the perimeter of the chip.

C. Computational Framework and Methodology

Our experiments are run on a Linux OS running on Intel Xeon 2.4 Ghz cores. For the placement portion of the algorithm there is no contention in the system. For the routing portion of the circuit, which includes a binary search to find the minimum channel width, all four cores are run in parallel to speed up this computationally heavy process. This has no impact on the results.

Finally, in combinatorial optimization experiments (including many CAD algorithms) the approaches are impacted by initial random seed. To eliminate some of the noise in these experiments we use a number of runs to average out the results. In the experiments, we use ten random seeds for each case and average these using the geometric mean, which is a normal procedure in CAD experiments. Any summary results across a range of benchmarks will also include a geometric mean.

VI. Results

In this section we show the results from two experiments. First, we show how the supergene GAs compare to GAs with the PMX crossover operator, the CSR crossover operator, and a GA with no crossover operator. In this experiment, the dominance factor of whether a supergene expresses itself is set to always on (meaning the supergene always expresses itself).

TABLE II. THE FPGA ARCHITECTURAL AND CAD PARAMETERS

	Architecture								CAD	
Parameter	**W**	N	K	F_{cin}	F_{cout}	F_s	routing	transistor sizing	timing factor	fixed I/O
Value	20% larger than minimum	10	5	0.18	0.1	3	uni-directional	27mwt	0.5	TRUE

The reason for this is, experimentally, we observed that super-clusters always benefit the GA. To observe the potential value of the dominance factor, the second experiment doubles the number of super-clusters by introducing false positive super-clusters, and the experiment shows how the supergenes evolve to deal with these false positives.

A. Algorithm Comparison with and without Supergene Mutations

In this experiment, we observe how each of the GAs compare with and without supergenes. Each GA has the same number of individuals in the population for each generation, the mutation rate is kept the same, and the number of generations is set to fifty. The only change in the algorithms is which crossover operator is used (PMX or CSR) on the fine-grain clusters, and if supergenes are used and mutated locally (coarse_i) or both locally and globally (coarse_ii).

Both Figures 3 and 4 show the normalized results for all of the algorithms compared to a GA that only uses a mutation operator and no crossover operator, and this is done to normalize the results. In both of these graphs, the x-axis includes each of the benchmarks and a geometrically averaged results of all benchmarks, and the y-axis is a normalized result compared against a GA with no crossover or supergene. The algorithms as described in the legend are:

- csr: represents the CSR crossover operator, supergenes, **No** global supergene mutations, and **No** local supergene mutations
- coarse_icsr: represents the CSR crossover operator, supergenes, global supergene mutations, and **No** local supergene mutations
- coarse_iicsr: represents the CSR crossover operator, supergenes, global supergene mutations, and local supergene mutations
- pmx: represents the PMX crossover operator, supergenes, **No** global supergene mutations, and **No** local supergene mutations
- coarse_ipmx: represents the PMX crossover operator, supergenes, global supergene mutations, and **No** local supergene mutations
- coarse_iipmx: represents the PMX crossover operator, supergenes, global supergene mutations, and local supergene mutations

In general, we observe that the channel width and critical path results are all improved by GA algorithms with crossover operators. There is one outlier in the channel width results for the coarse_icsr algorithm. Also, the GAs with supergene mutations at both the global and local level (coarse_ii) generate the best results compared to their respective crossover operators with the exception of the channel width results generated by the GA with PMX crossover (pmx). In this case, the pmx algorithm has better channel width results than all algorithms, but we hypothesize that this is the case of a trade-off that results in significantly worse critical path results.

The best result is a GA with supergenes mutated locally and globally has approximately 10% better critical path results and 4% better channel width results for the CSR crossover operators compared to a GA with CSR only. This result shows that the first ever medium-grain and fine-grain GA for FPGA placement improves previous GAs for placement.

B. Dominance Mutations in the Supergene

In the previous experiment, the mutation of the dominance factor (whether a supergene will express itself or not) was turned off. This was done based on experimental results that suggested that the super-clusters used in the above experiments, were always beneficial to keep. In this experiment, we increase the number of super-clusters by introducing potential false positives, and observe how many super-clusters are kept for the final best solution.

To introduce false positives, we do not simply introduce random super-clusters. Instead, we randomly select groupings (local placement) of clusters after SA has completed placement, and therefore, in theory, these super-clusters are reasonable super-clusters candidates, but they are not found as definitive good super-clusters. This set of good and possibly good super-clusters is what we use to evaluate if a supergenes dominance mutation will determine which supergenes to express.

TABLE III. RESULTS FOR AVERAGE NUMBER OF SUPERGENES EXPRESSED

Benchmark	# of total Supergenes	Supergenes Expressed	Extra Supergenes Expressed
cfc8	10	9.34	0.53
dconvert	78	72.16	7.37
desa	2	1.80	0.20
dsystemC	44	40.68	5.03
iir1	26	23.17	1.89
pajf	2	2.00	0.40
rsd1	10	9.31	0.51
rsd2	16	14.74	1.23
sv3	8	5.39	2.28
synth_14	22	9.52	1.33

Table III shows the results from the experiment. Column 1 shows the benchmark, and column 2 shows the total number of super-clusters (double the value of super-clusters in Table I). Columns 3 and 4 show the number of original supergenes and extra supergenes expressed in the final, best individual. Our hypothesis is that most of the original, good supergenes will be expressed, and the extra supergenes will not be expressed as the genome evolves. The results show that this is the case, and the dominance mutation in the supergene works well in removing false positives. In some cases, some of the false positives are expressed, and this suggests that there is either more work to be done on making a more sophisticated operator on dominance or that due to the fact that the GA is far from good quality that these anomalies are normal in the current optimization point that GAs are achieving for FPGA placement.

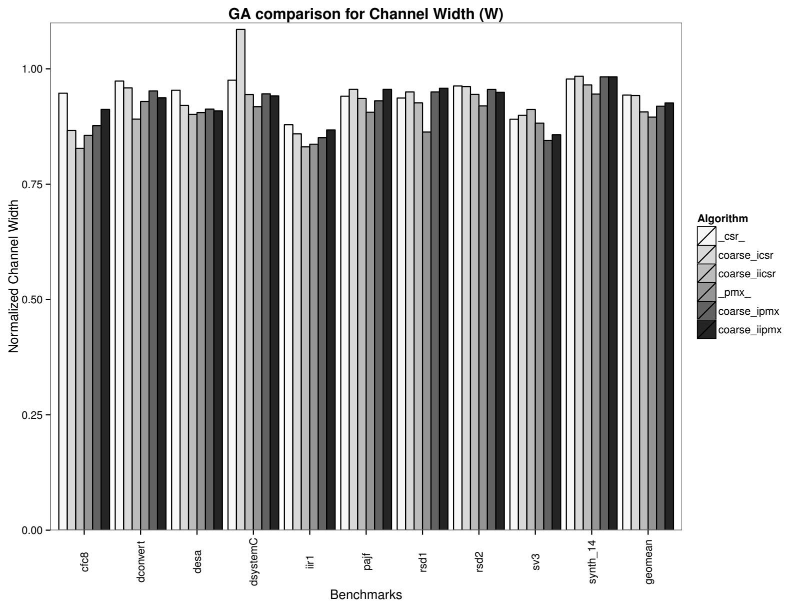


Fig. 3. A comparison of Channel Width results for all the GAs normalized against a mutation only GA

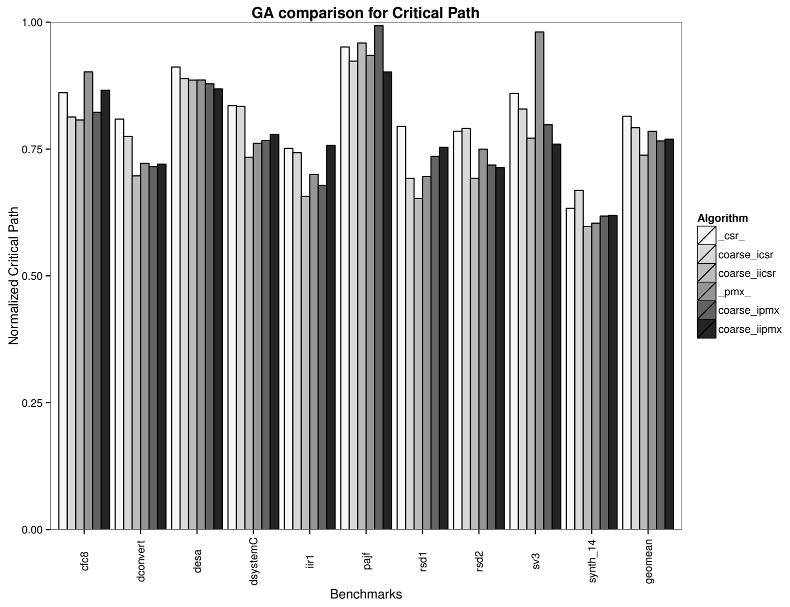


Fig. 4. A comparison of Critical Path results for all the GAs normalized against a mutation only GA

VII. Conclusions and Future Work

In this work, we introduced a GA for medium and fine-grain FPGA placement by including and experimenting with supergenes. Supergenes duplicate information in the genome, represent local improvements based on known information, and have the potential to be expressed instead of traditional genome information. We describe how supergenes can be used for the FPGA placement problem and the evolutionary operations performed on supergenes.

Our results show that supergenes improve the critical path results generated by a GA with CSR crossover operator by 10% over a GA without supergenes and the same crossover operator. We investigated the impact the different GAs had on channel width noting that there are some cases that supergenes did not improve the results, but this was not the case for the best GA with supergenes. Additionally, we investigated how mutating the dominance factor in a supergene impacts the expression of false positive super-clusters (not necessarily great supergene candidates), and this experiment showed that the GA kept the majority of good super-clusters and eliminated the expression of the majority of false positives.

Still GAs for FPGA placement are far from generating results as good as those generated by SA and other placement algorithms. The supergene improves the results, but there still lacks a crossover operator that can push GAs for FPGA placement forward and benefit from the easy parallelization of GAs. Therefore, concepts such as supergenes will make GAs better and GAs for FPGA placement better, but the majority of our future focus will be on finding better crossover functions within this application domain.

References

[1] A. Ludwin, V. Betz, and K. Padalia, “High-quality, deterministic parallel placement for fpgas on commodity hardware,” in *FPGA ’08: Proceedings of the 16th international ACM/SIGDA symposium on Field programmable gate arrays*, 2008, pp. 14–23. [Online]. Available: http://doi.acm.org/10.1145/1344671.1344676

[2] M. Joron, R. Papa, M. Beltrán, N. Chamberlain, J. Mavárez, S. Baxter, M. Abanto, E. Bermingham, S. Humphray, J. Rogers *et al.*, “A conserved supergene locus controls colour pattern diversity in heliconius butterflies,” *PLoS biology*, vol. 4, no. 10, p. e303, 2006.

[3] D. E. Goldberg and R. Lingle, Jr., “Alleles, loci, and the traveling salesman problem,” in *Proceedings of the 1st International Conference on Genetic Algorithms*, 1985, pp. 154–159. [Online]. Available: http://dl.acm.org/citation.cfm?id=645511.657095

[4] R. Collier, C. Fobel, G. Grewal, and M. Wineberg, “Depictions of genotypic space for evaluating the suitability of different recombination operators,” in *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, 2012, pp. 609–616. [Online]. Available: http://dl.acm.org/citation.cfm?id=2330250

[5] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1999.

[6] J. Rose, J. Luu, C. Yu, O. Densmore, J. Goeders, A. Somerville, K. Kent, P. Jamieson, and J. Anderson, “The vtr project: architecture and cad for fpgas from verilog to routing,” in *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*, 2012, pp. 77–86. [Online]. Available: http://dl.acm.org/citation.cfm?id=2145708

[7] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by Simulated Annealing,” *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.

[8] J. Kleinhans, G. Sigl, F. M. Johannes, and K. J. Antreich, “GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization,” *IEEE Transactions on Computer-Aided Design*, vol. 10, no. 3, pp. 356–365, Mar. 1991.

[9] D. Huang and A. Khang, “Partitioning-Based Standard-cell global placement with an Exact Objective,” in *International Symposium on Physical Design*, Napa Valley, CA, 1997, pp. 18–25.

[10] A. E. Dunlop and B. W. Kernighan, “A Procedure for Placement of Standard-Cell VLSI Circuits,” *IEEE Transactions on Computer-Aided Design*, vol. 4, no. 1, pp. 92–98, Jan. 1985.

[11] B. M. Riess and G. G. Ettelt, “Speed: Fast and Efficient Timing Driven Placement,” in *IEEE International Symposium on Circuits*, 1995, pp. 377–380.

[12] K. Vorwerk, A. Kennings, and A. Vannelli, “Engineering details of a stable force-directed placer,” in *ICCAD ’04: Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*, 2004, pp. 573–580.

[13] P. Mazumder and E. Rudnick, *Genetic Algorithms for VLSI Design, Layout & Test Automation*. Prentice Hall, 1999.

[14] R. Venkatraman and L. M. Patnaik, “An evolutionary approach to timing driven fpga placement,” in *GLSVLSI ’00: Proceedings of the 10th Great Lakes symposium on VLSI*, 2000, pp. 81–85. [Online]. Available: http://doi.acm.org/10.1145/330855.330986

[15] V. Betz and J. Rose, “Directional Bias and Non-Uniformity in FPGA Global Routing Architectures,” in *14th IEEE/ACM Int’l Conference on CAD*, 1996, pp. 652–659. [Online]. Available: http://portal.acm.org/citation.cfm?id=244522.244948

[16] Y. Meng, A. E. A. Almaini, and W. Pengjun, “Fpga placement optimization by two-step unified genetic algorithm and simulated annealing algorithm,” *Journal of Electronics (China)*, vol. 23, no. 4, pp. 632–636, 2007.

[17] P. Jamieson, “Revisiting genetic algorithms for the fpga placement problem,” in *GEM*, 2010, pp. 16–22. [Online]. Available: http://www.users.muohio.edu/jamiespa/html_papers/gem_10.pdf

[18] ——, “Exploring inevitable convergence for a genetic algorithm persistent fpga placer,” in *GEM*, 2011, pp. 1–8. [Online]. Available: http://www.users.muohio.edu/jamiespa/html_papers/gem_11.pdf

[19] R. Collier, C. Fobel, L. Richards, and G. Grewal, “A formal and empirical analysis of recombination for genetic algorithm-based approaches to the fpga placement problem,” in *Electrical & Computer Engineering (CCECE), 2012 25th IEEE Canadian Conference on*, 2012, pp. 1–6.

[20] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, 1st ed. Addison-Wesley Professional, January 1989. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20\&path=ASIN/0201157675

[21] D. Goldberg, B. Korb, and K. Deb, “Messy genetic algorithms: Motivation, analysis, and first results,” *Complex systems*, vol. 3, no. 5, pp. 493–530, 1989.

[22] J. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[23] A. Wu and I. Garibay, “The proportional genetic algorithm: Gene expression in a genetic algorithm,” *Genetic Programming and Evolvable Machines*, vol. 3, no. 2, pp. 157–192, 2002.

[24] P. Eggenberger, “Evolving morphologies of simulated 3d organisms based on differential gene expression,” in *Proceedings of the Fourth European Conference on Artificial Life*, 1997, pp. 205–213.

[25] *Stratix III Device Handbook*, Altera, 2006.

[26] *Virtex-5 Family Overview*, Xilinx, June 2006.

[27] P. Jamieson and J. Rose, “Enhancing the area efficiency of fpgas with hard circuits using shadow clusters,” *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 18, no. 12, pp. 1696 –1709, 2010. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5256139&tag=1

[28] D. Behrens, K. Harbich, and E. Barke, “Circuit partitioning using high level design information,” in *Proc. IDPT*, vol. 96, 1996, pp. 259–266.

[29] D. Singh and S. Brown, “Incremental placement for layout driven optimizations on fpgas,” in *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, 2002, pp. 752–759.

[30] L. Cheng and M. Wong, “Floorplan design for multimillion gate fpgas,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 25, no. 12, pp. 2795–2805, 2006.

[31] J. M. Emmert and D. Bhatia, “A methodology for fast fpga floorplanning,” in *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, 1999, pp. 47–56. [Online]. Available: http://doi.acm.org/10.1145/296399.296427

[32] J. Cohoon, S. Hegde, W. Martin, and D. Richards, “Distributed genetic algorithms for the floorplan design problem,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 10, no. 4, pp. 483–492, 1991.

[33] F. Gharibian, L. Shannon, and P. Jamieson, “Finding System-Level Information and Analyzing its Correlation to FPGA Placement,” in *20th International Conference on Field Programmable Logic and Applications*, 2010. [Online]. Available: www.users.muohio.edu/jamiespa/html_papers/FPL2010.pdf

[34] J. A. Roy, A. N. Ng, R. Aggarwal, V. Ramachandran, and I. L. Markov, “Solving modern mixed-size placement instances,” *Integration, the {VLSI} Journal*, vol. 42, no. 2, pp. 262 – 275, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167926008000461

[35] A. Wibowo and P. Jamieson, “Using simple ancestry to deter inbreeding for persistent genetic algorithm search,” in *GEM*, 2012, pp. 23–29. [Online]. Available: http://www.users.muohio.edu/jamiespa/html_papers/gem_12.pdf

[36] R. Collier and M. Wineberg, “The problems with counting ancestors in a simple genetic algorithm,” in *Proceedings of the 9th European conference on Advances in artificial life*, ser. ECAL’07, 2007, pp. 855–864. [Online]. Available: http://dl.acm.org/citation.cfm?id=1771390.1771489

[37] F. Glover, “Future paths for integer programming and links to artificial intelligence,” *Comput. Oper. Res.*, vol. 13, pp. 533–549, May 1986. [Online]. Available: http://dl.acm.org/citation.cfm?id=15310.15311

[38] L. J. Eshelman, “The chc adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination,” in *FOGA*, 1990, pp. 265–283.