Chapter 1
Speeding-Up Expensive Evaluations in High-Level Synthesis Using Solution Modeling and Fitness Inheritance

Christian Pilato, Daniele Loiacono, Antonino Tumeo, Fabrizio Ferrandi, Pier Luca Lanzi, Donatella Sciuto
Dipartimento di Elettronica ed Informazione — Politecnico di Milano
{pilato,loiacono,tumeo,ferrandi,lanzi,sciuto}@elet.polimi.it

Abstract — High-Level Synthesis (HLS) is the process of developing digital circuits from behavioral specifications. It involves three interdependent and NP-complete optimization problems: (i) the *operation scheduling*, (ii) the *resource allocation*, and (iii) the *controller synthesis*. Evolutionary Algorithms have been already effectively applied to HLS to find good solution in presence of conflicting design objectives. In this paper, we present an evolutionary approach to HLS that extends previous works in three respects: (i) we exploit the NSGA-II, a multi-objective genetic algorithm, to fully automate the design space exploration without the need of any human intervention, (ii) we replace the expensive evaluation process of candidate solutions with a quite accurate regression model, and (iii) we reduce the number of evaluations with a fitness inheritance scheme. We tested our approach on several benchmark problems. Our results suggest that all the enhancements introduced improve the overall performance of the evolutionary search.

1.1 Introduction

High-Level Synthesis (HLS) [8] is concerned with the design and implementation of digital circuits starting from a behavioral description, a set of goals and constraints, and a library of different types of resources. HLS typically consists of three steps: the scheduling, the resource allocation and the controller synthesis. The scheduling assigns each operation to one or more clock cycles (or control steps) for the execution. The resource allocation assigns the operations and the produced values to the hardware components and interconnects them using connection elements. Finally, the controller synthesis

provides the logic to issue datapath operations, based on the control flow. Unfortunately, it is non-trivial to solve these problems as they are NP-complete and strongly interdependent. In addition, the high-level synthesis problem is multi-objective and most of the design objectives are contrasting by nature. Therefore, developers usually apply an iterative refinement cycle: at each step they (i) manually apply transformations, (ii) synthesize the design, (iii) examine the results coming from the synthesis of the design solution, and (iv) modify the design to trade off the design objectives. This process is usually called *design space exploration*.

Evolutionary algorithms (EAs) have been successfully applied [12, 21] to such complex explorations, since their behavior is very similar to the one of a designer: they iteratively improve a set of solutions (i.e. alternative designs) using the results of their evaluations as a feedback to guide the search in the solution space. In addition, EAs proved to work well on large optimization problems even if (i) the search space is constrained, (ii) there are few information available on EAs can also easily deal with different objectives, without the need of combining them into a single objective function. The main drawback of EA approaches is the need to evaluate a huge number of design alternatives. This is a serious concern as in HLS problems the solution evaluation is a very expensive process. To meet the time-to-market constraints we need to shorten the design process without reducing the quality of the solutions discovered.

In this paper we present an evolutionary framework to perform a fully automated design space exploration for HLS problems. In addition, to compute the fitness of the evolved solution we replace the usual expensive evaluation process with a cost model coupled with an inheritance fitness scheme. In particular, our approach extends previous works on the application of EAs to HLS [14, 21, 24, 25] basically in three respects: (i) while in previous works focused on evolutionary approaches to optimize a human designed objective function, we exploit *NSGA-II* [9], a multi-objective genetic algorithm, to perform a fully automated design space exploration; (ii) we exploit a regression model to perform a fast and quite accurate evaluation of the candidate solutions; (iii) to our knowledge, this is the first work that applied a fitness inheritance scheme to HLS in order to reduce the number of evaluations. We validated our approach on several benchmark problems. Our empirical results suggest that both the regression model introduced and the fitness inheritance scheme result in an improvements of the design space exploration process.

This chapter is organized as follows. After discussing relevant work in Section 1.2, we describe our approach in Section 1.3. In Section 1.4 we discuss the issues of the solution evaluation. Then, in Section 1.5 we show how the fitness can be computed through a prediction model to decrease dramatically the cost , consists of replacing a part of the expensive solution evaluation with an estimation model that, given some relevant features of the solution, provides an estimation of its objective values. Then, we present two different techniques to reduce expensive evaluations: cost modeling and fitness in-

heritance [29, 34]. The former technique, presented in The latter technique, detailed in Section 1.6, allows to reduce the number of fitness evaluations by replacing some of them with a surrogate, based on the fitness values of other individuals previously evaluated. Experimental evidences on a set of historical benchmarks for the HLS problem, both in terms of quality of the solutions w.r.t. the design objectives and overall execution time of the exploration, are presented and discussed for each technique.

1.2 Related Work

The common techniques used in high-level synthesis can be classified into three categories: *exact*, *heuristic* and *non-deterministic approaches*.

The *exact approaches* [7, 18] exploit mathematical formulations of the problem and may find the optimal solution. Unfortunately, their computational requirements grow exponentially with the size of the problem and are impractical for large designs.

The *heuristic approaches* [8, 28, 35] work on a single operation or resource at once and perform continuous refinements on the set of solutions. The decision process is deterministic, so they do not explore all the design alternatives, possibly leading to sub-optimal solutions. Furthermore, most of these techniques perform the scheduling and the allocation sub-tasks separately, with the scheduling usually performed as the first step.

To support scalability and to explore a larger set of alternative designs, several *non-deterministic approaches* (e.g., [20]), and in particular GAs [12, 14, 21, 24, 25], have been efficiently applied to HLS. Most of them focused on only one of the HLS sub-task. In [24], GAs are used to schedule the operation, while in [25] they are used to allocate and bind a scheduled graph. Grewal et al. [14] implemented a hierarchical genetic algorithm, where genetic module allocation is followed by a genetic scheduling. Araújo et al. [1] used a genetic programming approach, where solutions are represented as tree productions (rephrased rules in the hardware description language grammar) to directly create Structured Function description Language (SFL) programs. This work presents a different approach w.r.t. the previous ones, but it is difficult to control the optimizations. Krishnan and Katkoori [21] proposed a priority-based encoding, where solutions are represented as a list of priorities that defines in which order the operations should be chosen by the scheduling algorithm. However, they performed a single-objective optimization using a weighted average of the design objectives, that has been proved to be not effective [39]. Several works [12, 25] introduced a binding-based encoding, also for system-level synthesis [27, 36], where solutions are represented as the binding between operations and functional units where they will be executed. In some of these approaches the exploration can generate

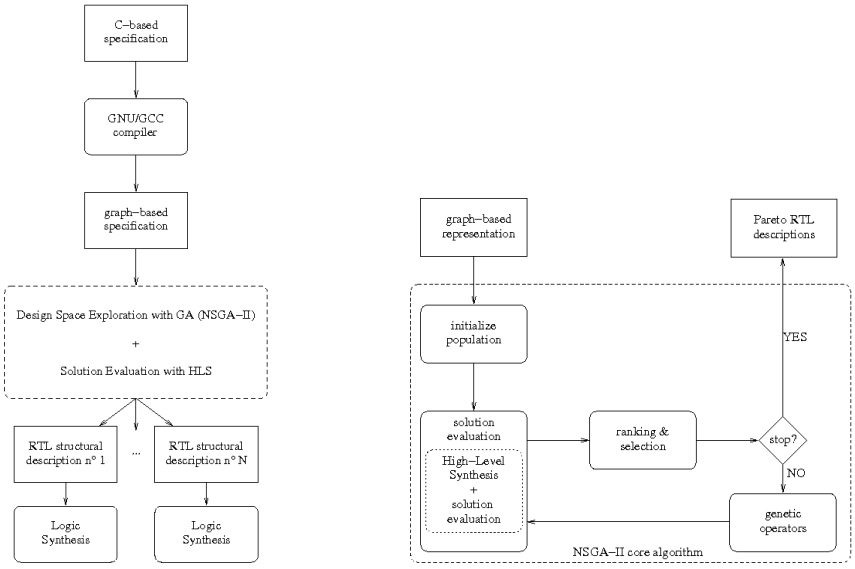
unfeasible solutions that have to be recovered or discarded, wasting time and computation resources.

EAs often require to evaluate a number of candidate solutions that might easily result computationally unfeasible. This generally happens in real-world problem and it is also the case of HLS. Accordingly, in the literature several *evaluation relaxation* [13] techniques have been introduced to speedup EAs: an accurate, but computationally expensive, fitness function is replaced by a less accurate, but inexpensive, surrogate. Following the early empirical design, theories have been developed to understand the effect of approximate surrogate functions on population sizing and convergence time and to enhance speedups (see [31] for further details). The surrogate can be either endogenous [34] or exogenous [2, 19, 23]. *Fitness inheritance* [34] is one of the most promising endogenous approach to evaluation relaxation: the fitness of some proportion of individuals in the population is inherited from the parents. Sastry et al. [33] use a model based on least squares fitting, applied in particular to extended compact genetic algorithm (eCGA [16]). Chen et al. [6] present their studies on fitness inheritance in multi-objective optimization as a weighted average of parent fitness, decomposed in the different n objectives. Recent studies investigated the impact of fitness inheritance on real-world applications [11] and different exploration algorithms [30]. Exogenous surrogate are typically used in engineering applications [2, 10] and consists of developing a simplified model of the real problem to provide an inexpensive surrogate of the fitness function. In particular, in HLS several simplified models for area and timing have been proposed in the literature. In [26], simple metrics are proposed to drive the optimization algorithms, even if some elements are not correctly considered (e.g., steering logic or effects of optimizations performed by the logic synthesis tools). In [3] the area is estimated with a linear regression approach that is also able to model the effects of the logic optimizations. Unfortunately, most of the models proposed provide a poor guidance to the optimization process as they do not take into account the resource binding and the interconnections [5]. In this work we focus on data-flow applications that involve only area models, however we refer the interested reader to [4, 22] for timing estimation models.

1.3 Design Space Exploration with Multi-Objective Evolutionary Computation

The proposed methodology is shown in Figure 1.1(a). The inputs are the behavioral description of the problem in C language, a library of resource descriptions and a set of constraints to be met, specified in XML format. We exploit a customized interface to the GNU *GCC* compiler[1] to generate

[1] GCC - GNU Compiler Collection, *http://gcc.gnu.org*



(a) Overall Methodology (b) Design Space Exploration

Fig. 1.1 Methodology overview: squared boxes represent intermediate representations, rounded boxes represent tools. The dashed box represents the methodology described in this chapter.

the related GIMPLE, representing the behavioral specification. From this, a combined control and data dependencies data structure (*CDFG*) is built. CDFG allows the identification of the operations that should be mapped and scheduled on the functional units described in the resource library provided as input, as well as of the precedences among them.

The core of our methodology, shown in Figure 1.1(b) and detailed in Section 1.3.1, is the design space exploration that exploits a multi-objective GA (*NSGA-II* [9]) to concurrently optimize the different design objectives: the area and the latency. This design space exploration iteratively improves a set of candidate solutions to explore the most promising design subspaces. Finally, a Register-Transfer Level (RTL) specification in a hardware description language (e.g. VHDL, Verilog or SystemC) is generated for each one of the non-dominated solutions contained into the final population resulting from the exploration algorithm.

1.3.1 Design Space Exploration Core

The design space exploration core is shown in Figure 1.1(b). Initially a *population* of N candidate solutions is randomly created and evaluated through

a complete high-level synthesis, with respect to the design objectives. In the current implementation, area and performance have been considered. Once the evaluation is completed, the solutions are sorted. After the initialization step, a new population of N candidate solutions is created. In particular, each element of the new population, called *offspring*, is generated by applying the common genetic operators (i.e., *crossover* and *mutation*) to the existing solutions, called *parents*. Finally, each offspring created is evaluated as well and added to the population. The resulting population of size $2N$ is then sorted again and the worst N solutions are discarded. All the steps described above, except the initialization, are thus iteratively repeated until the following stopping criterion is met. Whenever the set of best solutions is not improved in the last 10 iterations, the size of the population, N, is increased by 50%. When, even increasing the population size, no best solutions are found, the optimization process is stopped. At the end, the non-dominated solutions found by the exploration algorithm are returned.

1.3.2 Genetic Algorithm Design

In this section, we present the design of the NSGA-II that drives our design space exploration.

Solution encoding. In this methodology, the chromosome is simply a vector where each gene describes the mapping between the related operation in the behavioral specification and the functional unit where it will be executed. With this formulation both the resource allocation (i.e., the total number of required functional units) and the operations binding (i.e., the assignment of the operations to the available units) are encoded at the same time and all the information that is necessary to generate the structural description of the design solution is encoded. This encoding was introduced for the first time in [12] and is inspired to the approach proposed in [25]. The main advantage in using this encoding is that all genetic operators (see Section 1.3.2) create feasible solutions. In fact, the recombination of the operations binding simply results in a new allocation or binding. In this way, good solutions can be obtained just using common genetic operators, without needing procedures to recover unfeasible solutions.

Initial population. At the beginning of each run, an initial population of admissible resource bindings is created. It can be created by random generation or, to cover a larger design space and to speedup the exploration process, by generating some known solutions (e.g. the one with the minimum number of functional units or the minimum latency). This allows the algorithm to start from some interesting points and then to explore around to improve them.

Fitness function. To evaluate the solutions we used the following multi-objective fitness function:

$$F(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \end{bmatrix} = \begin{bmatrix} Area(x) \\ Time(x) \end{bmatrix} \tag{1.1}$$

where $Area(x)$ is an estimation of the area occupied by the solution x, $Time(x)$ is an estimation of the latency of the solution x, computed as the worst case execution time of the scheduled solution, in terms of clock cycles. The goal of the genetic algorithm is to find the best trade-offs with respect to this cost function.

Ranking and selection. The ranking of solutions is an iterative process. At iteration k, all the solutions are first sorted according to the *fast-non-dominated-sort*. Then, the non-dominated solutions are classified as solution at the k-level and removed from the solutions to be ranked. The process is repeated until all the solutions have been ranked. At the end of the evolutionary process, the whole set of solutions ranked as the best ones will be the outcome of the optimization. We refer to this set as the aprroximation of the *Pareto-optimal* set discovered by the evolutionary process.

Genetic operators. To explore and exploit the design space, the usual genetic operators are used, the *unary mutation* and the *binary crossover*. The two operators are applied respectively with probability P_m and P_c. *Mutation* is an operator used for finding new points in the search space. Mutation has been implemented with a relatively low rate (e.g., P_m=10%) and it is applied as follows: each gene is modified with probability P_μ, changing the corresponding binding information. *Crossover* is a reproduction technique that mates two parent chromosomes and produces two offspring chromosomes. Given two chromosomes, a standard *single-point* crossover is applied with a high probability (e.g., P_c=90%). The crossover mechanism mixes the binding information of the two parent solutions.

1.3.3 Performance Measure

The outcome of muti-objective EAs is a set of solutions that represent the best estimate of the *Pareto front* in the objective space. Accordingly, evaluating and comparing the outcome of different EAs is not trivial as it is in single-objective optimization. In particular, several metrics have been introduced in the literature [38] with different features and aims. In general a performance metric can provide either a relative measure (e.g., Non Dominated Combined Set Ratio [37]) or an absolute measure (e.g., $\mathcal{S}$ metric [38]). The former type of metric are devised to compare only two set of solutions, while the latter allow to rank several set of solutions on a specific problem. In this work we used a performance metric that is equivalent to the $\mathcal{S}$ metric as (i) we

need to compare several set of solutions and (ii) it is a scale-independent metric. In minimization problems with two objectives the $\mathcal{S}$ metric can be computed as the hypervolume between the set and the anti-ideal objective vector [17]. Unfortunately in HLS, the anti-ideal objective vector is not always defined. Accordingly we set each objective of the anti-ideal vector to the worst value discovered during all the evolutionary runs. In addition, for a better readability we defined the performance measure as the area *complementary* to the $\mathcal{S}$ metric in the positive quadrant. Accordingly, the smaller is the used metric, the better the set of solutions is.

1.4 Solution Evaluation

The crucial point to obtain a fast and effective convergence of the exploration is the quality of the solution evaluation. In particular, the values of the fitness function should be as close as possible to the effective values that would be obtained through the actual implementation of the design on the target technology and the evaluation of the desired characteristics (e.g., area or latency). For this reason, the best fitness is obtained with a complete synthesis of each design solution. A complete synthesis includes the following two steps. First, a high-level synthesis flow is performed from a fully specified design solution to a structural description of the circuit. Then a logic synthesis step is applied to generate the circuit for the target technology from its structural description.

Our approach targets the Field Programmable Gate Array (FPGA) technology. A FPGA is a semiconductor device that can be configured by the customer or the designer after manufacturing. FPGAs are becoming an interesting alternative to Application Specific Integrated Circuits (ASICs) as they allow to customize the system without the need of an expensive development process. In particular, FPGAs fit the needs of embedded systems design where they are used to develop accelerators specific to improve the performance of the applications that will be used on the systems. Nevertheless, the choice of this technology introduces additional difficulties in the design process that tools for high level synthesis and design space exploration need to address. FPGAs are composed by a set of configurable logic units, typically Look Up Tables (LUTs) with four inputs and one output, that are used to represent the logic functions, and a series of flip-flops. These elements are organized in Configurable Logic Blocks (CLBs) that communicates through a programmable interconnection network. Modern FPGAs may also feature dedicated blocks for some type of operations (e.g. hardware multipliers) and embedded memories. The generation of a FPGA based design requires to process the specification of the circuit in a hardware description language with a synthesizer, like Integrated Software Environment (ISE) for Xilinx devices or Quartus for ALTERA solutions. A FPGA synthesis tool follows

several stages. The first stage (synthesis) transforms the specification into a set of logic primitives and memory elements for the reconfigurable device. The second stage (mapping) maps these basic components to the specific device available. In the last stage (routing) the blocks are connected together and with the input/output pins. The initial data on the occupation are available after the first stage. This process, however, is quite expensive in terms of time. Depending on the complexity of the design, a single logic synthesis may require hours to be completed. As a result, the solution cannot be evaluated by simply adding the contribution of the allocated components, but the effects of the logic synthesis step has to be somehow considered. In previous works, only the area of the functional units (i.e. the resources that performs the operations) and registers were included in the solution evaluation, as they were considered much more relevant than interconnection elements (e.g., multiplexers). However, recent studies [5, 15] demonstrated that the area of the interconnection elements has by far outweighed the area of the functional units. In ASICs, this brings undesirable side-effects, like an unacceptable propagation due to the long wires determined by an inefficient components placement. In FPGAs, this situation is critical for area calculation, since a large amount of LUTs may be used to connect and wire the functional blocks. This strongly motivates the design of techniques that take into account the amount and size of interconnection elements. Not considering them could lead to an inaccurate area estimation and to a final solution that does not meet the area constraints.

Benchmark	**HLS time (s)**	**Logic Synthesis time (s)**	**Total time (s)**
arf	0.35	100.72	101.07
bandpass	0.99	28.50	29.49
chemical	0.37	122.03	122.40
dct	1.02	133.70	134.72
dct wang	1.04	124.45	125.49
dist	0.96	248.54	249.50
ewf	0.39	121.35	121.74
fir	0.07	43.61	43.80
paulin	0.06	32.19	32.25
pr1	0.84	121.70	122.54
pr2	1.19	176.08	177.27
tseng	0.05	14.00	14.05
Avg.	0.61	105.57	106.18

Table 1.1 Examples of computational effort for the complete synthesis of common benchmarks for high-level synthesis.

Unfortunately, due to the complexity of analysis and the interdependence of the synthesis steps, all the information is available only after the complete synthesis of the design solutions. Some examples of the computational effort required to produce a complete synthesis for various designs with our HLS tool and the Xilinx ISE version 10.1 are reported in Table 1.1. We used a system with a Intel Core 2 Duo T7500 CPU (2,2 GHz, 4 MB of second level cache) and 2 GB of memory. These results clearly show that that the com-

plete synthesis cannot be efficiently included into any black-box optimization algorithm that usually performs a huge number of design evaluations.

This motivates us to investigate different solutions to reduce the execution time of the solution evaluation, limiting the impact on the quality of the final solutions. If we reduce the time required to evaluate a design solution, more alternatives could be analyzed in the same time and a larger portion of the design space can be explored.

In the following sections we introduce two different techniques to speed-up the evaluation process. In Section 1.5 we discuss how to replace the fitness computation with a cost model to avoid the expensive logic synthesis step to evaluate candidate solutions. In particular, we show how the accuracy of the cost model affect the overall performance. Then, in Section 1.6 we investigate the application of a *fitness inheritance* inheritance mechanism to reduce the number of evaluations performed without degrading the performance of the evolutionary process.

1.5 Building Cost Models

In this section, the time-consuming logic synthesis step is substituted with a model of performance and area, based on relevant features of the structural descriptions obtained by the high-level synthesis step. To compute the performance, it is necessary to count the control steps required by the design to execute all the operations, which correspond to the number of clock cycles required to execute the design. To compute the area, then, it is necessary to perform the logic synthesis of the specifications produced by the HLS flow. As described in Section 1.2, the typical approach in literature is to build a fitness surrogate that, considering some features of the design, is able to estimate its occupation. In the following, we present two possible cost models for the area: one linearly combines the number of functional units present in the design and their area and counts the memory elements, the other one is a linear regression that also takes into account interconnections.

However, even if the solution modeling allows to reduce the time required to evaluate a solution, it introduces an approximation that could affect the explorations. For this reason, the accuracy of the models and the quality of the designs obtained by the exploration using these models will be discussed and analyzed, respectively, in Section 1.5.2.1 and Section 1.5.2.2.

1.5.1 Cost Models

One of the simplest models used in HLS flows counts the number of functional units and memory elements. An area estimation in terms of LUTs

$$\#A.Area.FF = \sum_{R \in A.DataPath.Registers} sizeof(R) + log_2(A.FSM.NumControlStates)$$

$$\#LUT = \sum_{F \in A.DataPath.FunctionalUnits} F.Area$$

$$A.Area = \#A.Area.FF + \#LUT$$

Fig. 1.2 Simplified model to estimate area occupation for the structural design A.

for each type of functional unit (e.g. adder, subtractors, multipliers) can be easily obtained through the synthesis of such elements. The linear combinations of these values provides an initial estimation of the overall occupation of the design [21]. The HLS flow can instead estimate the number of single bit flip-flops by counting the number of registers required by the design, for both the data-path and the state encoding registers of the control-FSM. Consequently, the first estimation model we adopted to compute the area of a design is shown in Fig.1.2. This model is easy to develop and allows a very fast estimation of the area occupation of the design. However, it can only model how the exploration affects the number of functional units or registers, and does not account for the effects of the interconnection elements. The contribution of the controller is limited to the number of memory elements required to encode the state. The logic to compute the outputs or the transition function is ignored. Such a solution was proposed, several years ago, mainly for data-intensive designs targeting ASIC technology, considering that the interconnections and the controller had a reduced impact on these design.

Nevertheless, as discussed in Section 1.4, recent studies demonstrated that this approach is not applicable with FPGAs [5], and it is becoming inefficient also for ASICs [15]. We thus investigated more detailed models for generating the required values to verify if it is possible to obtain better approximations. We started with an already existing area model for FPGAs [3], and generalized it for several reasons. First, the vendors (e.g., Xilinx or Altera) offer tools with different approaches to translate the structural descriptions into the logic functions and to interconnect the logic blocks. Second, the devices, even if provided by the same vendor, can use different architectures (e.g., LUTs with a different number of inputs). So, we introduced a generic model and a methodology to specialize it, in order to address different vendors' tools and different devices. The final area model we used for fast estimation is shown in Figure 1.3. For each architecture A the model divides the area into two main parts: the Flip-Flop part and the LUT part. While the Flip-Flop part is easy to estimate using the same formula of the previous approach, the LUT part is a little more complex. Four main parts contribute to the global area in terms of LUT: FU, FSM, MUX and Glue. The FU part corresponds to the contribution of the functional units and so its value is still the sum of the area value of each functional unit. The other three parts (FSM, MUX, Glue) are obtained by using a regression-based approach:

$$\#FF_{FSM} = \lceil \alpha_1 * log_2(A.FSM.NumControlStates) + \beta_1 \rceil$$
$$\#FF_{DataPath} = \sum_{R \in A.DataPath.Registers} \alpha_2 * sizeof(R) + \beta_2$$
$$A.Area.FF = \alpha_3 * \#FF_{FSM} + \beta_3 * \#FF_{DataPath}$$
$$\#LUT_{FSM} = \lceil \alpha_4 * A.FSM.NumControlStates + \beta_4 * A.FSM.Inputs + \gamma_4 * A.FSM.Outputs + \delta_4 \rceil$$
$$\#LUT_{FU} = \sum_{F \in A.DataPath.FunctionalUnits} \alpha_5 * F.Area + \beta_5$$
$$\#LUT_{MUX} = \sum_{M \in A.DataPath.Mux} \lceil \alpha_6 * M.Input + \beta_6 * sizeof(M) + \gamma_6 \rceil$$
$$\#LUT_{Glue} = \lceil \alpha_7 * \#LUT_{FSM} + \beta_7 * A.DataPath.NumRegisters + \gamma_7 \rceil$$
$$A.Area.LUT = \alpha_8 * \#LUT_{FSM} + \beta_8 * \#LUT_{FU} + \gamma_8 * \#LUT_{MUX} + \delta_8 * \#LUT_{Glue} + \epsilon_8$$
$$A.Area = \alpha_8 * A.Area.LUT + \alpha_9 * A.Area.FF$$

Fig. 1.3 Linear regression model to estimate area occupation for the structural design A.

- the FSM contribution is due to the combinatorial logic used to compute the output and next state;
- the MUX contribution is due to the number and size of multiplexers used in the datapath;
- the Glue contribution is due to the logic to enable writing in the flip flops and to the logic used for the interaction between the controller and the datapath.

The model is then specialized for the particular vendor's tools and devices by using a linear regression approach similar to [3], obtaining an accurate estimation of the design objectives, if properly adapted. For this reason, one of the main drawbacks is that, each time the designer changes the experimental setup, it requires an initial phase of tuning, that could be time-consuming and error-prone.

1.5.2 Experimental Evaluation

In this section, the models are validated by using the set of benchmarks presented in [7] and targeting a Virtex XC2VP30 FPGA. The logic synthesis is executed with Xilinx ISE ver. 10.1. We performed the coefficient extraction for the model based on linear regression and its validation using two datasets, each one composed by different hardware architectures of the benchmarks. The resulting model is shown in Fig. 1.4.

The error of the two models is discussed in Section 1.5.2.1, while their impact on the final estimates of the Pareto-optimal set is analyzed in Section 1.5.2.2. In particular, we demonstrate which model is better to drive the optimization process carried on by the genetic algorithm.

$$\#FF_{FSM} = \lceil log_2(A.FSM.NumControlStates) \rceil$$

$$\#FF_{DataPath} = \sum_{R \in A.DataPath.Registers} sizeof(R)$$

$$A.Area.FF = \#FF_{FSM} + \#FF_{DataPath}$$

$$\#LUT_{FSM} = \lceil 1.99 * A.FSM.NumControlStates - 0.24 * A.FSM.Inputs + 1.50 * A.FSM.Outputs - 9.97 \rceil$$

$$\#LUT_{FU} = \sum_{F \in A.DataPath.FunctionalUnits} F.Area$$

$$\#LUT_{MUX} = \sum_{M \in A.DataPath.Mux} \lceil (0.79 * sizeof(M)) \rceil$$

$$\#LUT_{Glue} = \lceil 0.7 * \#LUT_{FSM} + 1.10 * A.DataPath.NumRegisters \rceil$$

$$A.Area.LUT = \#LUT_{FSM} + \#LUT_{FU} + \#LUT_{MUX} + \#LUT_{Glue}$$

$$A.Area = A.Area.LUT + A.Area.FF$$

Fig. 1.4 Model used to estimate area occupation for the FPGA design A using Xilinx ISE ver. 10.1. and targeting a Virtex XC2VP30 FPGA device.

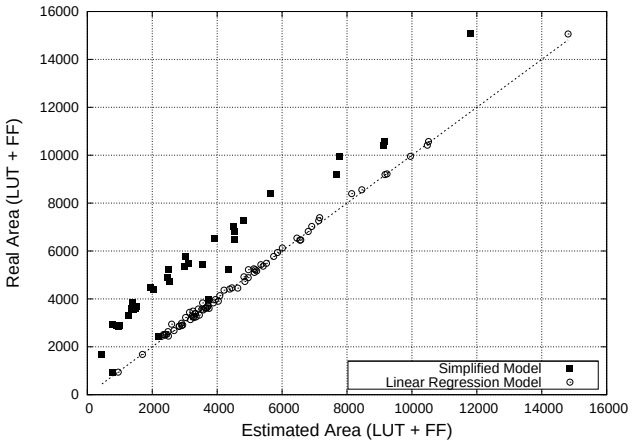


Fig. 1.5 Validation of the FPGA area models.

1.5.2.1 Accuracy of Models

Figure 1.5 presents the validation for the models described in the previous section. The dashed line represents the ideal situation, where the estimated values are equal to the real ones, obtained with an actual synthesis on the target device. Squared dots represent the values associated to the first model, where only functional units and registers are considered. Round dots, instead, represent the values obtained with the linear regression model. We validated the models on a dataset composed by 73 designs that represent different architectures of the benchmarks described in [7] and shown in Table 1.1.

The plot shows that the simplified model systematically underestimates the real values. This happens because the contribution due to multiplexers and steering logic is not considered. On the other hand, the model based

Benchmark	Simplified			Linear Regression		
	Area	#Pareto Points		Area	#Pareto Points	
		NSGA-II DSE	Synthesis		NSGA-II DSE	Synthesis
arf	63,010	7	6	**60,341**	9	9
dct	111,392	8	6	**107,808**	14	14
dct wang	115,167	11	9	**110,198**	16	16
dist	158,716	14	13	**157,049**	20	20
ewf	73,969	10	9	**72,634**	13	13
pr1	72,990	9	8	**70,978**	12	12
pr2	162,130	17	14	**154,503**	19	19

Table 1.2 Comparison of the models applied to a set of benchmarks.

on linear regression approximates the real values with a good accuracy. In particular, the simplified model shows an average error of 43.39±20.00%, while the maximum error is 73.35%. The model based on linear regression, instead, has an average error equal to 2.22±2.20%, with a maximum error of 11.85%. Thus, we can confirm that is able to accurately estimate all the area contributions of a structural description and that it can be effectively integrated in the proposed methodology to drive the exploration algorithm.

1.5.2.2 Performance of the Methodology

The error information is insufficient to determine which model should be preferred. We need to evaluate the effects of the adoption of the models on the resulting estimates of the Pareto-optimal set. The more accurate is the model, the better it would drive the design space exploration, resuling in a better estimate of the Pareto-optimal set. However, even a simple model might be enough to perform an effective design space exploration, if it would be able to identify and consider the most relevant features of the design.

Consequently, we performed different experiments, alternatively adopting different area models. Each experiment consists of 100 generations and involves a population of 100 candidate solutions. The results averaged over 10 runs are shown in Table 1.2, where the column *Area* measures the quality of the non-dominated set discovered. In particular, the lower is this value, the better is the outcome of the optimization processes. *NSGA-II DSE* and *Synthesis* values represent, respectively, the Pareto points coming out from the exploration algorithm and the results after their actual synthesis. The results show that the linear regression model systematically outperforms the simplified model. The reason is that the linear regression model is more accurate, and it is able to consider the effects on the solution evaluation of all the components contained in the final architecture. Furthermore, having a model that generates a more accurate fitness function results in a larger number of points in the estimate of the Pareto-optimal set.

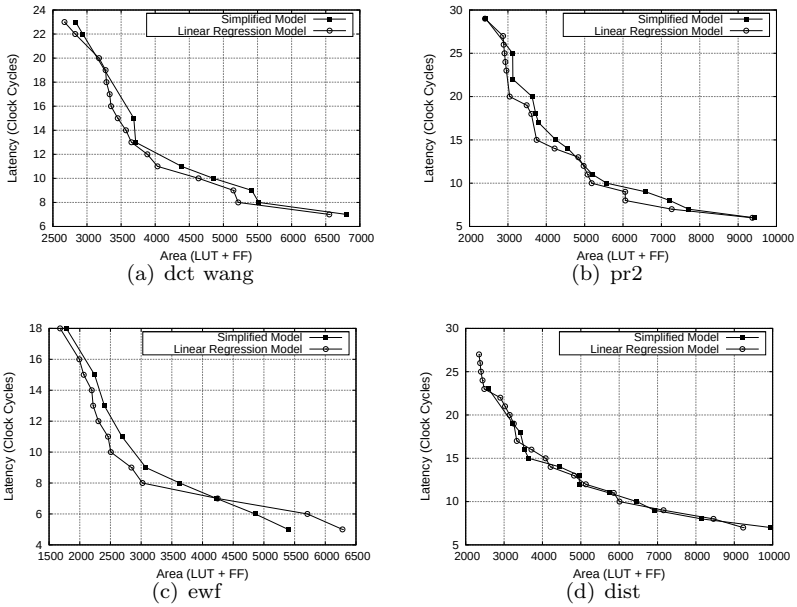


Fig. 1.6 Examples of comparison of obtained Pareto curves using the different models.

Some interesting approximations of the Pareto-optimal curve are also graphically compared in Fig. 1.6. In Fig. 6(a) and Fig. 6(b) we see that the linear regression model systematically outperforms the simplified one in terms of quality of the Pareto-optimal set. In Fig. 6(c), for large designs, the model that considers only functional units and registers obtains better results. In fact, in this region of the design space, the impact of the multiplexer is limited (about 15-20%) and a fitness function focused only on functional components and registers is more suitable to drive the exploration algorithm. In the other region of the space, where few functional units are in the designs, the multiplexers have a larger impact (about 70-75%) and the fitness function that takes into account their occupation obtains better results. Finally, in Fig. 6(d), the multiplexers are not so relevant for the design. As a result the two models are almost equivalent, as also shown by the similar values in Table 1.2.

1.6 Fitness Inheritance

Table 1.1 shows that also the HLS step is computationally intensive, even if much less than the logic synthesis one. HLS impact becomes bigger as the dimension of the problem grows. We thus expect that, when applied to larger problems, it could become another significant bottleneck of the methodology.

For this reason, in this section we exploit *fitness inheritance* to substitute all the steps of the complete synthesis by interpolating the fitness of previously evaluated individuals. *Fitness inheritance* is a technique totally orthogonal to the solution proposed in the previous section. The individuals used for fitness inheritance can, in fact, be evaluated with any approach (e.g., actual synthesis or modeling with one of the proposed models). The key idea is that, with this approach, we try to limit the overall number evaluations rather than reducing the time required for a single evaluation (i.e. the synthesis steps, HLS or logic synthesis). Interpolation is usually much less time consuming, thus we can save some of the time required for a complete synthesis.

Note that this technique is less dependent on the problem than solution modeling. In fact, to build the model, the designer should identify the relevant features of the design solutions, synthesize the related hardware descriptions and establish a correspondence. On the contrary, fitness inheritance is only based on the definition of the chromosome encoding and the fitness of previously evaluated individuals.

However, to produce an effective surrogate, we needed to carefully take into account some aspects. In particular, we focused our attention on the percentage of individuals to be estimated, on the parents to choose and on how to combine their fitness. We present and discuss these aspects in Section 1.6.1, and then compare the quality of some different solutions in Section 1.6.2.

Provided a proper analysis of these aspects, the results show that fitness inheritance is able to consistently reduce the execution time of all the methodology. We also demonstrate that, if the parameters are not correct, the method can even degrade rather than improving the performance of the exploration algorithm.

1.6.1 Inheritance Model

In the proposed approach, only in the first, initial population the fitness of all the individuals is evaluated. In the subsequent populations, only the fitness of a portion of the population is evaluated, while the remaining ones inherit the fitness through interpolation of the values already computed. In particular, the fitness of individual Ind_i is inherited with probability p_i. To compute the fitness estimation for Ind_i, we need to calculate the distance between it and all the individuals that can be effectively used for the estimations. The estimation can be based on the ancestors, i.e., all the individuals that have been effectively evaluated starting from the first generation, or on the parents, i.e., all the individuals that have been effectively evaluated only in the latest generation. In both the cases, we will call this set S in the rest of the section. The fitness value of Ind_i is thus estimated as follows. The chromosome of Ind_i is mapped onto a binary vector of size N, where each

variable of the vector is uniquely related to a gene of the chromosome. The vector is instantiated by the following delta function:

$$\delta_{i,j}^{k} = \begin{cases} 0 \; if \; Ind_i[k] = Ind_j[k] \\ 1 \; otherwise \end{cases} \tag{1.2}$$

where $Ind_i[k]$ is the value associated to the *k-th* gene of the individual Ind_i. After the delta function has been computed for all the N genes of the chromosome, the distance $d_{i,j}$ between individual Ind_i and individual Ind_j is calculated as follows:

$$d_{i,j} = \frac{\sum_{k=1}^{N} \delta_{i,j}^{k}}{N} \tag{1.3}$$

this function is normalized with the size of chromosome, so its value is always between 0 and 1. The distance $d_{i,j}$ measures the similarity of two individuals. If these are totally different (there is not any matching gene), the value will be 1. On the other hand, if the two individuals are identical, the value will be 0. Only individuals that are considered *neighbors* in this space will be kept for the fitness estimation. We call r the maximum distance that an individual should have to be kept. The name r is used to remember the term *radius*, since the region delimited by this value can be imagined as a N-dimensional hypersphere centered at individual Ind_i. All the individuals $Ind_j \in S$, having distance smaller than the radius r, can be considered as points inside this hypersphere. Therefore, all these individuals will be considered for estimation and the distance value is modified as follows:

$$d'_{i,j} = \begin{cases} d_{i,j} \; if \; d_{i,j} \leq r \\ 1 \quad if \; d_{i,j} > r \end{cases} \tag{1.4}$$

where all individuals outside the hypersphere are equivalent to points at infinite distance and they will not be considered for estimation. To perform the estimation, we require a minimum number of points in this region. If there are not enough points, it means that there is no sufficient local information to estimate and individual. So, it will be really evaluated. If there are enough points, instead, the estimation can be performed on the set S' of points, selected as follows:

$$Fit_i^z = \frac{\sum_{j \in S'} f(Fit_j^z) * g(1 - d'_{i,j})}{\sum_{j \in S'} g(1 - d'_{i,j})} \tag{1.5}$$

for each objective z. Fit_k^z is the value of the objective z for the individual Ind_k and $(1 - d'_{i,j})$ is used as a measure of closeness between individuals. f and g are functions that change the contribution of the two terms. We formulated the term $(1 - d'_{i,j})$ in this way since the distance $d'_{i,j}$ does not go to *infinite*, but has a value between 0 and 1. Therefore, we consider the values associated to 1 equivalent to an *infinite* distance (i.e., no contribute to the fitness). As explained above, this weighted average is computed for all the

Benchmarks	Model	w/o inheritance #Eval.	w/o inheritance Exec. Time(s)	Ancestors #Eval.	Ancestors Exec. Time (s)	Ancestors diff (%)	Parents #Eval.	Parents Exec. Time(s)	Parents diff (%)
arf	linear	9,639	1,064.65	4,721	1,211.68	+13.81%	7,567	888.66	-16.53%
	quadratic			5,048	1,327.19	+24.65%	8,160	801.86	-24.68%
	exponential			5,330	1,374.83	+29.13%	9,027	985.59	-7.42%
dct	linear	11,150	3,677.55	7,263	5,074.26	+37.98%	7,758	2,699.86	-26.58%
	quadratic			7,758	5,492.04	+49.34%	7,131	2,360.44	-35.81%
	exponential			7,319	5,135.87	+39.65%	7,867	2,625.69	-28.60%
dct wang	linear	10,837	3,470.16	6,945	4,906.20	+41.38%	7,348	2,385.12	-37.27%
	quadratic			7,479	5,456.37	+57,24%	7,758	2,549.29	-26.53%
	exponential			6,160	4,083.51	+17,68%	7,312	2,689.09	-22.51%
dist	linear	12,683	3,907.81	8,315	6,181.55	+58.18%	7,812	3,402.46	-12.93%
	quadratic			7,607	5,254.36	+34.46%	7,358	3,048.29	-21.99%
	exponential			8,376	5,995.74	+53.43%	7,801	3,590.28	-8.13%
ewf	linear	9,575	1,165.55	6,218	2,074.99	+78.03%	6,518	814.60	-30.11%
	quadratic			6,392	2,127.26	+82.51%	64,18	790.00	-32.22%
	exponential			6,256	2,095.66	+79.80%	6,578	889.94	-23.65%
pr1	linear	9,773	2,542.35	8,358	5,514.68	+116.91%	7,548	2,058.11	-19.05%
	quadratic			6,834	3,879.37	+52.59%	6,912	1,878.11	-26.13%
	exponential			6,681	3,594.25	+41.48%	7,154	1,958.15	-22.98%
pr2	linear	10,610	4,044.71	6,423	4,718.46	+16.66%	6,958	3,589.25	-11.26%
	quadratic			6,930	5,086.70	+25.76%	7,198	3,578.02	-11.54%
	exponential			6,937	5,119.57	+26.57%	7,277	3,547.66	-12.59%
					Avg.	+46.53%		Avg.	-21.82%
					Std. Dev.	25.90%		Std. Dev.	8.81%

Table 1.3 Comparison of the weighting functions about fitness evaluations and execution time.

objectives considered in the optimization. The resulting value is then returned to the genetic algorithm, which can so proceed. A flag is also associated to the individual Ind_i to remember that the fitness has been estimated and not really evaluated. This allows the algorithm to identify the estimated individuals when needed.

In particular, in the last generation the fitness of all the individuals are tested for evaluation. Individuals that have already been evaluated will be skipped, while the estimated individuals will be effectively evaluated. Thus, when the exploration ends, all the individuals on which the final non-dominated set is computed will have a real fitness value associated.

1.6.2 Experimental Evaluation

In this section, we evaluate different aspects related to fitness inheritance and compare several parameter settings. The parameters for the GA are the same used in Section 1.5.2.2. In all the experiment, the fitness evaluation uses the linear regression model. In Section 1.6.2.1 we present, discuss, and compare different functions to weight the fitness contributions of the evaluated individuals. In Section 1.6.2.2 we apply fitness inheritance both to the ancestors and to the parents and compare the results. Finally, we analyze the effects of different inheritance percentages (p_i) and distance rates (r).

Benchmarks	Model	w/o inheritance Area	w/o inheritance #Pareto	Ancestors Area	Ancestors #Pareto DSE	Ancestors #Pareto Synth.	Parents Area	Parents #Pareto DSE	Parents #Pareto Synth.
arf	*linear*	63,157	9	63,756	9	9	63,633	11	11
	quadratic			63,633	11	11	**62,729**	12	12
	exponential			65,097	9	9	64,941	12	12
dct	*linear*	113,526	14	113,151	14	14	113,732	12	12
	quadratic			**111,598**	11	10	113,732	12	12
	exponential			114,550	13	11	113,469	10	10
dct wang	*linear*	112,868	16	113,351	12	12	112,782	15	15
	quadratic			114,778	17	17	**112,484**	14	14
	exponential			114,389	13	13	113,911	14	14
dist	*linear*	169,487	20	168,955	18	17	170,731	19	19
	quadratic			171,706	18	17	170,578	18	18
	exponential			169,708	21	20	**167,900**	19	19
ewf	*linear*	72,634	13	76,946	14	12	**73,245**	13	13
	quadratic			75,503	13	12	74,366	11	11
	exponential			77,000	13	13	74,184	13	13
pr1	*linear*	75,405	12	76,286	11	10	75,168	11	11
	quadratic			76,878	11	10	75,083	11	11
	exponential			76,580	12	11	**75,000**	12	12
pr2	linear	156,906	19	158,110	19	18	154,903	18	18
	quadratic			161,812	21	19	**154,186**	19	19
	exponential			162,195	22	20	160,800	20	20

Table 1.4 Comparison of the weighting functions about quality of the results.

1.6.2.1 Weighting Functions

We considered three weighting functions (i.e., function g in Eq.1.5) for inheritance: linear, quadratic and exponential. The first model is computed as follows:

$$Fit_i^z = \frac{\sum_{j \in S'} Fit_j^z * (1 - d'_{i,j})}{\sum_{j \in S'} (1 - d'_{i,j})} \tag{1.6}$$

where the fitness of the evaluated individuals are linearly combined with the related distances $1 - d'_{i,j}$ from the candidate individual Ind_i. While, the second model is computed as follows:

$$Fit_i^z = \frac{\sum_{j \in S'} Fit_j^z * (1 - d'_{i,j})^2}{\sum_{j \in S'} (1 - d'_{i,j})^2} \tag{1.7}$$

where the quadratic function in $(1 - d'_{i,j})$ is used to increase the weight of distance, similarly to the Physics equations for gravity or magnetism. However, we adopt a proportion with $(1-d)^2$ and not $(1/d)^2$, that allows dealing with *infinite* distance as described above. The last model is computed as:

$$Fit_i^z = \frac{\sum_{j \in S'} Fit_j^z * (e^{1-d'_{i,j}} - 1)}{\sum_{j \in S'} (e^{1-d'_{i,j}} - 1)} \tag{1.8}$$

where the distance is exponentially weighted, emphasizing even more the contribution of the nearest individuals to the fitness estimation of Ind_i. These functions have been applied both to the ancestors and to the parents. The

distance rate has been set to $r = 0.20$ and the inheritance rate to $p_i = 0.5$. In the former case, the set S of individuals considered increases generation by generation, while, in the latter case, the size is constant and related to the size of the population. When the ancestors are used, the inheritance model analyzes all the elements of the set for distance calculation, and the time required for fitness inheritance could overcome the time required by the function evaluation itself. Thus, in this case, fitness inheritance reduces the number of evaluations, but may also degrade the overall execution time of the methodology. At opposite, if the methodology is applied only to the parents, both the number of evaluations and the execution time of the methodology are significantly reduced. Since less individuals are available for computing the inheritance information (see Eq. 1.4), the number of evaluations is larger than with the ancestors. Table 1.3 shows the data about the number of evaluations and about the overall execution time.

Finally, Table 1.4 compares the quality of the results with the different weighting functions. As in Section 1.5.2.2, the area delimited by the approximated Pareto-optimal curve gives a qualitative evaluation of the explorations. The results show that the quadratic function is the most efficient solution to weight the fitness contributions. In fact, this function emphasizes the individuals closer to the candidate more than the linear function. With respect to the exponential function, which (strongly) emphasize only very similar individuals, it also consider more distant contributions (always inside the radius).

1.6.2.2 Parameter Analysis

In this section, different inheritance rates (p_i) and different distance rates (r) are studied. The parameters for the GA are the same used in Section 1.5.2.2. The fitness evaluation uses the linear regression model and exploits inheritance on *parents* with the *quadratic* weighting function in all the experiments.

Table 1.5 shows the results of explorations where fitness inheritance is applied with different inheritance rates. Note that values of p_i between 0.40 and 0.55 provides a good trade-off between the quality of the exploration and the related execution time. The reason is that, with lower values, few individuals are chosen for inheritance. On the other hand, with higher values, the number of really evaluated individuals is limited. When there are not enough similar individuals (at least 10), we swap the fitness evaluation to the HLS flow and the area model. Therefore, the execution time is not reduced as expected. The results obtained in our experiments are also consistent with the optimal proportion for inheritance derived in [32], defined as follows:

$$0.54 \leq p_i* \leq 0.558 \tag{1.9}$$

Benchmarks	p_i	w/o inheritance Area	#Pareto	#Eval.	Exec. Time(s)	w inheritance Area	#Pareto DSE	#Pareto Synth.	#Eval.	Exec. Time(s)	diff (%)
arf	0.20	63,157	9	9,639	1,064.65	64,088	12	12	**7,681**	**834.00**	-21.66%
	0.30					**62,724**	11	11	8,852	961.30	-9.71%
	0.40					62,990	11	11	8,812	981.43	-7.82%
	0.50					63,239	11	11	8,259	913.65	-14.18%
	0.55					62,820	10	10	8,160	888.66	-16.53%
	0.60					64,275	12	12	8,702	965.97	-9.27%
	0.70					63,654	11	11	8,655	961.38	-9.70%
dct	0.20	113,526	14	11,150	3,677.55	113,487	14	14	8,158	3,248.20	-11.67%
	0.30					113,909	16	16	7,789	2,874.11	-21.85%
	0.40					113,104	15	15	7,441	2,581.47	-29.80%
	0.50					113,732	12	12	7,131	2,360.44	-35.81%
	0.55					112,223	11	11	**7,062**	**2,236.98**	-39.17%
	0.60					114,051	16	16	7,325	2,514.02	-31.64%
	0.70					**111,080**	12	12	7,587	2,636.42	-28.31%
dct wang	0.20	112,868	16	10,837	3,470.16	113,952	13	13	8,258	3,025.20	-12.82%
	0.30					**111,487**	14	14	8,126	2,854.01	-17.76%
	0.40					113,536	15	15	7,887	2,741.36	-21.00%
	0.50					112,484	14	14	7,758	**2,549.29**	-26.54%
	0.55					114,283	15	15	**7,747**	2,569.10	-25.97%
	0.60					112,842	17	17	7,854	2,698.47	-22.24%
	0.70					113,706	16	16	7,981	2,747.11	-20.84%
dist	0.20	169,487	20	12,683	3,907.81	166,060	17	17	7,414	3,658.10	-6.39%
	0.30					**161,432**	17	17	7,401	3,698.43	-5.36%
	0.40					167,804	18	18	**7,333**	3,154.01	-19.29%
	0.50					170,578	18	18	7,358	**3,048.29**	-21.99%
	0.55					167,801	17	17	7,441	3,341.22	-14.50%
	0.60					167,472	19	19	7,551	3,418.99	-12.51%
	0.70					170,151	16	16	7,547	3,507.67	-10.24%
ewf	0.20	**72,634**	13	9,575	1,165.55	74,623	13	13	9,147	847.10	-27.32%
	0.30					74,143	12	12	7,765	858.36	-26.36%
	0.40					73,609	13	13	7,010	802.19	-31.17%
	0.50					74,366	11	11	6,418	790.00	-32.22%
	0.55					74,053	11	11	**6,211**	**767.41**	-34.16%
	0.60					73,234	12	12	6,478	789.23	-32.29%
	0.70					73,023	13	13	6,441	796.59	-31.66%
pr1	0.20	75,405	12	9,773	2,542.35	75,308	13	13	8,012	2,236.39	-12.03%
	0.30					**74,309**	9	9	7,477	2,056.47	-19.11%
	0.40					75,319	10	10	7,087	1,969.78	-22.52%
	0.50					75,083	11	11	6,912	1,878.11	-26.13%
	0.55					74,888	9	9	**6,898**	**1,789.56**	-29.61%
	0.60					75,045	10	10	7,101	1,941.02	-23.65%
	0.70					74,831	10	10	7,011	1,867.53	-26.54%
pr2	0.20	156,906	19	10,610	4,044.71	160,628	20	20	8,101	3,856.12	-4.66%
	0.30					**150,630**	22	22	7,812	3,785.54	-6.41%
	0.40					158,903	21	21	7,485	3,696.36	-8.61%
	0.50					154,186	19	19	7,198	3,578.02	-11.54%
	0.55					154,241	15	15	**7,012**	3,547.10	-12.30%
	0.60					155,714	21	21	7,025	3,423.11	-15.37%
	0.70					159,150	21	21	6,894	**3,326.98**	-17.74%
										Avg.	-20.43%
										Std. Dev.	9.13%

Table 1.5 Comparison of different inheritance rate about quality of the exploration, fitness evaluations and execution time.

Benchmarks	r	w/o inheritance Area	w/o inheritance #Pareto	w inheritance Area	w inheritance #Pareto DSE	w inheritance #Pareto Synth.
arf	0.10	63,157	9	63,549	10	10
	0.20			63,626	11	11
	0.25			63,307	11	11
	0.50			**62,906**	11	11
dct	0.10	113,526	14	**112,626**	14	14
	0.20			113,116	13	13
	0.25			112,630	17	17
	0.50			113,234	14	14
dct wang	0.10	112,868	16	113,347	13	13
	0.20			115,027	15	15
	0.25			**111,391**	17	17
	0.50			112,979	13	13
dist	0.10	169,487	20	171,159	19	19
	0.20			168,305	16	16
	0.25			**168,195**	17	17
	0.50			169,162	19	19
ewf	0.10	72,634	13	73,156	14	14
	0.20			73,302	12	12
	0.25			**71,693**	11	11
	0.50			75,009	13	13
pr1	0.10	75,405	12	76,607	13	13
	0.20			**75,372**	11	11
	0.25			76,214	12	12
	0.50			77,654	12	12
pr2	0.10	156,906	19	157,005	19	19
	0.20			154,814	18	18
	0.25			158,718	21	21
	0.50			**153,866**	18	18

Table 1.6 Comparison of different distance rate about quality of the results.

Finally, Table 1.6 reports the results obtained with $p_i = 0.5$ while changing the distance rates r. Almost all the considered rates give good results. However, values comprised between 0.20 and 0.25 perform best. In fact, with lower values, limited information is available for inheritance, while, with higher values, additional noise is introduced in the interpolation.

1.7 Conclusions

In this work, we presented an evolutionary approach to HLS design space exploration problem based on NSGA-II, a multi-objective evolutionary algorithm. We exploited two orthogonal techniques, surrogate fitness and fitness inheritance, to reduce the time necessary to the expensive solution evaluations. The fitness surrogate was computed with a linear regression model that takes into account the contributions of all the components of the design (e.g., interconnections or glue logic) and the effect of the optimizations introduced by the logic tool: replacing the logic synthesis process with such a surrogate model, we can save a lot of computational time. Fitness inheritance was used to reduce the number of evaluations, by evaluating only a fixed portion of the population. We validated our approach on several benchmarks and our

results suggest that both the proposed techniques allows to speed-up the evolutionary search without degrading its performance. At the best of our knowledge, this is the first framework for the HLS design space exploration that exploits at the same time a surrogate fitness model as well as a fitness inheritance scheme.

References

1. Araújo SG, Mesquita AC, Pedroza A (2003) Optimized Datapath Design by Evolutionary Computation. In: IWSOC: International Workshop on System-on-Chip for Real-Time Applications, pp 6–9
2. Barthelemy JFM, Haftka RT (1993) Approximation concepts for optimum structural design - a review. Structural Optimization (5):129–144
3. Brandolese C, Fornaciari W, Salice F (2004) An area estimation methodology for FPGA based designs at SystemC-level. In: DAC: Design Automation Conference, ACM, New York, NY, USA, pp 129–132
4. Chaiyakul V, Wu ACH, Gajski DD (1992) Timing models for high-level synthesis. In: EURO-DAC '92: European Design Automation Conference, IEEE Computer Society Press, Los Alamitos, CA, USA, pp 60–65
5. Chen D, Cong J (2004) Register binding and port assignment for multiplexer optimization. In: ASP-DAC: Asia South Pacific Design Automation Conference, pp 68–73
6. Chen JH, Goldberg DE, Ho SY, Sastry K (2002) Fitness inheritance in multi-objective optimization. In: GECCO: Genetic and Evolutionary Computation Conference, pp 319–326
7. Cordone R, Ferrandi F, Santambrogio MD, Palermo G, Sciuto D (2006) Using speculative computation and parallelizing techniques to improve scheduling of control based designs. In: ASPDAC: Asia South Pacific Design Automation Conference, ACM, Yokohama, Japan, pp 898–904
8. De Micheli G (1994) Synthesis and Optimization of Digital Circuits. McGraw-Hill
9. Deb K, Agrawal S, Pratab A, Meyarivan T (2000) A Fast and Elitist Multi-Objective Genetic Algorithm: NSGA-II. In: PPSN: Parallel Problem Solving from Nature Conference, Springer. Lecture Notes in Computer Science No. 1917, pp 849–858
10. Dennis J, Torczon V (1997) Managing approximate models in optimization. In: Alexandrov N, Hussani M (eds) Multidisciplinary design optimization: State-of-the-art, SIAM, pp 330–347
11. Ducheyne E, Baets BD, Wulf RD (2003) Is fitness inheritance useful for real-world applications?
12. Ferrandi F, Lanzi PL, Palermo G, Pilato C, Sciuto D, Tumeo A (2007) An evolutionary approach to area-time optimization of FPGA designs. In:

ICSAMOS: International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, pp 145–152
13. Grefenstette JJ, Fitzpatrick JM (1985) Genetic search with approximate function evaluation. In: International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA, pp 112–120
14. Grewal G, O'Cleirigh M, Wineberg M (2003) An evolutionary approach to behavioural-level synthesis. In: CEC: IEEE Congress on Evolutionary Computation, ACM Press, New York, NY, USA, 8-12, pp 264–272
15. Gu Z, Wang J, Dick RP, , Zhou H (2007) Unified incremental physical-level and high-level synthesis. IEEE Trans on CAD of Integrated Circuits and Systems 26(9):1576–1588
16. Harik G (1999) Linkage Learning via Probabilistic Modeling in the ECGA
17. Huband S, Hingston P (2003) An evolution strategy with probabilistic mutation for multi-objective optimisation. In: Evolutionary Computation, 2003. CEC 2003. IEEE Congress on, IEEE Press, Piscataway NJ, pp 2284–2291
18. Hwang CT, Leea JH, Hsu YC (1991) A formal approach to the scheduling problem in high level synthesis. IEEE Trans on CAD of Integrated Circuits and Systems vol. 10(4):464–475
19. Jin Y (2005) A comprehensive survey of fitness approximation in evolutionary computation. Soft Comput 9(1):3–12
20. Kollig P, Al-Hashimi B (1997) Simultaneous scheduling, allocation and binding in high level synthesis. Electronics Letters 33(18):1516–1518
21. Krishnan V, Katkoori S (2006) A genetic algorithm for the design space exploration of datapaths during high-level synthesis. IEEE Trans Evolutionary Computation 10(3):213–229
22. Kuehlmann A, Bergamaschi RA (1992) Timing analysis in high-level synthesis. In: ICCAD: International Conference on Computer-Aided Design, IEEE Computer Society Press, Los Alamitos, CA, USA, pp 349–354
23. Llorà X, Sastry K, Goldberg DE, Gupta A, Lakshmi L (2005) Combating user fatigue in iGAs: partial ordering, support vector machines, and synthetic fitness. In: GECCO: Conference on Genetic and evolutionary computation, ACM Press, New York, NY, USA, pp 1363–1370
24. Mandal C, Chakrabarti PP, Ghose S (1996) Design space exploration for data path synthesis. International Conf on VLSI Design pp 166–170
25. Mandal C, Chakrabarti PP, Ghose S (2000) GABIND: a GA approach to allocation and binding for the high-level synthesis of data paths. IEEE Transaction on Very Large Scale Integration System 8(6):747–750
26. Meribout M, Motomura M (2004) Efficient metrics and high-level synthesis for dynamically reconfigurable logic. IEEE Trans Very Large Scale Integr Syst 12(6):603–621
27. Palesi M, Givargis T (2002) Multi-objective design space exploration using genetic algorithms. In: CODES: International Symposium on Hardware/software Codesign, ACM, New York, NY, USA, pp 67–72

28. Paulin PG, Knight JP (1989) Force-directed scheduling for the behavioral synthesis of ASICs. IEEE Trans on CAD of Integrated Circuits and Systems vol. 8(n. 6):661–679
29. Pilato C, Palermo G, Tumeo A, Ferrandi F, Sciuto D, Lanzi PL (2007) Fitness inheritance in evolutionary and multi-objective high-level synthesis. In: IEEE Congress on Evolutionary Computation, pp 3459–3466
30. Reyes-Sierra M, Coello C (2005) A study of fitness inheritance and approximation techniques for multi-objective particle swarm optimization. vol 1, pp 65–72 Vol.1
31. Sastry K (2001) Evaluation-relaxation schemes for genetic and evolutionary algorithms. Master's thesis, General Engineering Department, University of Illinois at Urbana-Champaign, Urbana, IL.
32. Sastry K, Goldberg DE, Pelikan M (2001) Don't evaluate, inherit. In: GECCO: Genetic and Evolutionary Computation Conference, Morgan Kaufmann, San Francisco, California, USA, pp 551–558
33. Sastry K, Lima CF, Goldberg DE (2006) Evaluation relaxation using substructural information and linear estimation. In: GECCO 2006, ACM, Seattle, USA, pp 419–426
34. Smith RE, Dike BA, Stegmann SA (1995) Fitness inheritance in genetic algorithms. In: SAC: Symposium on Applied computing, ACM Press, New York, NY, USA, pp 345–350
35. Stok L (1994) Data Path Synthesis. Integration, the VLSI Journal vol. 18(1):1–71
36. Teich J, Blickle T, Thiele L (1997) An evolutionary approach to system-level synthesis. In: CODES Workshop, p 167
37. Wanner E, Guimaraes F, Takahashi R, Fleming P (2006) A quadratic approximation-based local search procedure for multiobjective genetic algorithms. In: Evolutionary Computation, 2006. CEC 2006. IEEE Congress on, pp 938–945, DOI 10.1109/CEC.2006.1688411
38. Zitzler E, Optimization M, Zrich EH, Thiele L, Dr P, Dr P, Deb K (1999) Evolutionary algorithms for multiobjective optimization: Methods and applications. PhD thesis
39. Zitzler E, Deb K, Thiele L (2000) Comparison of multiobjective evolutionary algorithms: Empirical results. Evolutionary Computation 8(2):173–195