IET Research Journals

Submission Template for IET Research Journal Papers

IET Journals
The Institution of Engineering and Technology

ISSN 1751-8644
doi: 0000000000
www.ietdl.org

A Novel Multi-objective Optimisation Algorithm for Routability and Timing Driven Circuit Clustering on FPGAs

Yuan Wang[2], *Martin A. Trefzer*[1], *Simon J. Bale*[1], *James A. Walker*[1], *Andy M. Tyrrell*[1],

[1] *Department of Electronic Engineering, University of York, UK*
[2] *School of Engineering and Technology, University of Hertfordshire, UK*
* *E-mail: y.wang54@herts.ac.uk, martin.trefzer@york.ac.uk*

Abstract:
Circuit clustering algorithms fit synthesised circuits into FPGA configurable logic blocks (CLBs) efficiently. This fundamental process in FPGA CAD flow directly impacts both effort required and performance achievable in subsequent place-and-route processes. Circuit clustering is limited by hardware constraints of specific target architectures. Hence, better circuit clustering approaches are essential for improving device utilisation whilst at the same time optimising circuit performance parameters such as, e.g., power and delay. In this paper, we present a method based on multi-objective genetic algorithm (MOGA) to facilitate circuit clustering. We address a number of challenges including CLB input bandwidth constraints, improvement of CLB utilisation, minimisation of interconnects between CLBs. Our new approach has been validated using the "Golden 20" MCNC benchmark circuits that are regularly used in FPGA-related literature. The results show that the method proposed in this paper achieves improvements of up to 50% in clustering, routability and timing when compared to state-of-the-art approaches including VPack, T-VPack, RPack, DPack, HDPack, MOPack and iRAC. Key contribution of this work is a flexible EDA flow that can incorporate numerous objectives required to successfully tackle real-world circuit design on FPGA, providing device utilisation at increased design performance.

1 Introduction

Field Programmable Gate Arrays (FPGAs) have developed significantly over the past 20+ years and are the number one choice for prototyping complex digital designs. Their flexibility moves them into many application areas such as reconfigurable computing, evolvable hardware and fault-tolerant systems. However, this flexibility puts limitations on maximum achievable design speed and area on FPGA. The resource utilisation is thereby not constrained solely by the hardware structure, but also depends significantly on the application mapping process. Such computer-aided design (CAD) flow for a implementing digital circuits on FPGA consists of synthesising a design into gate-level netlists whose components are mapped onto basic logic elements of the fabric. These are then clustered into higher-level configurable logic blocks (CLBs). Hence, this process is known as circuit clustering [1–3]. Separate placement and routing stages then attempt to map clustered functions onto the fabric and connect these blocks to form the circuit for a given application. Clustering is therefore a fundamental process in CAD flow that is linked to the architecture of a specific FPGA and therefore possibly limited by a variety of hardware constraints. Hence, better circuit clustering approaches are essential to the successful and effective use of FPGAs.

Clustering a large netlist, the synthesised circuit, into groups is a non-trival task when considering all clustered circuit properties. The problem becomes more complex as the inherent hardware constraints are considered as well. When a group of logic elements has been selected and clustered into a logic block, the circuit properties that have to be optimised are often conflicting, and it is usually a non-trivial problem to optimally balance multiple clustering objectives. This cannot be efficiently addressed by simply weighting and accumulating them into a single performance metric or figure of merit. Circuit clustering is by its very nature a complex multi-objective optimisation problem. In this paper, we therefore propose a multi-objective circuit clustering technique to solve and optimise the circuit-clustering stage in CAD for FPGA design.

The method proposed in this paper is divided into two steps, each using multi-objective generic algorithms (MOGAs): the first generates initial solutions using predictive metrics, and the second generates specific optimised solutions. Both steps consider five optimisation objectives using non-dominated sorting and crowding-distance selection based on NSGA-II [4]. The MOGAs described in this paper produce multiple unique solutions, which are based on Pareto optimality. This method gives maximum flexibility and makes it possible to add additional clustering metrics later without changing the core algorithm or invalidating solutions found already. The MCNC-20 benchmark suite is used to test and validate the proposed method. We compare the experimental results to other state-of-the-art FPGA circuit clustering methods, VPack [1], T-VPack [5], RPack [6], DPack, HDPack [7], MOPack [8] and iRAC [3]. The improvements achieved with the proposed method are up to 4.33% in reducing the number of logic blocks and up to 14.24% in reducing interconnect when compared to iRAC, the best-performing circuit clustering method which also enhances both FPGA area usage and routability. The most important result of our method is that optimised solutions achieve a speed up of mapped circuits by up to 27.62% compared to T-VPack and outperforms other well-known methods. The results indicate that the MOGA-based method can efficiently solve the FPGA circuit clustering problem taking both routability and delay into account.

2 Circuit Clustering in FPGA Design Flow

This section provides an overview of the FPGA architecture and state-of-the-art FPGA design flow. In order to implement a complete design onto FPGA, many design objectives have to be considered by automated CAD tools. Fig. 1(a) shows a cluster-based FPGA CAD flow.

2.1 FPGA Model

A cluster-based island-style FPGA is widely used, and its logic and routing blocks are arranged in a 2D mesh [9]. To lower the routing difficulties and improve application performance, a logic block, also known as configurable logic block (CLB), usually contains N basic logic elements (BLEs) and internal routing resources [10]. The BLE is the smallest configurable logic element that includes a k-input look-up table (LUT) and a reconfigurable flip-flop (FF). An important feature is usually presented in many Altera FPGAs [11, 12], which is the input-bandwidth constraint [13]. This means that the total number of inputs, I, of a CLB is less than $N \times k$, due to resource limitations. Though this constraint does not exist in all modern FPGAs, it represents an additional issue and has been the subject of research in circuit clustering. Additionally, the FPGA CAD research tool VPR [14] used here to facilitate design mapping is also based on an input-bandwidth-constraint CLB model. In most FPGA circuit clustering literature, I, N and k are normally set to 18, 8 and 4 respectively, and CLBs are assumed to have a unique clock [15]. These assumptions are also used in this paper.

The routing architecture is based on a X row by Y column array of CLBs. Each CLB is connected to the routing channel, wire segments, via the input and output connection blocks [16]. The connectivity of the input and output connection blocks are defined by two parameters, $F_{c,in}$ and $F_{c,out}$, the fraction of wire segment width in the channel (which refers to the pre-defined channel width W) to the connection number of the input or output of the CLB is used [9]. The switch block contains a set of programmable routing switches, and is positioned between CLBs. Switch block flexibility is defined by the parameter F_s representing the number of possible connections that a wire segment can make to other wire segments.

2.2 Circuit Clustering for FPGAs

Circuit clustering, also known as circuit packing, is a process to partition a synthesised circuit into sub circuits to enable mapping them to FPGA without breaking any CLB hardware constraints. Circuit clustering also indicates how to best group (pair) the BLEs based on their LUT and FF connections. Fig. 1(b) illustrates this process.

Circuit clustering is a fundamental process in the CAD flow, and the quality of clustering can significantly impact subsequent placement and routing processes which then directly affect the performance of the circuit. It can be significantly more problematic—or even impossible—to optimise a circuit's performance in subsequent steps of the CAD flow if clustering is ineffective. When clustering BLEs into CLBs, even if two solutions have the same number of clustered CLBs, their BLE combinations within the CLBs can be different. Therefore, circuit clustering is a complex grouping problem similar to multi-objective bin packing—a well-known NP-hard problem [17], but with additional constraints and requirements.

2.3 Multi-objective Problem Formulation

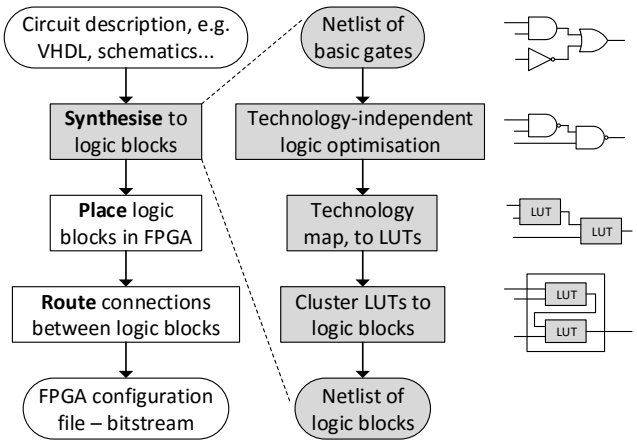
The basic requirements of circuit clustering which refer to routability-driven circuit clustering are: Firstly, it is required to cluster all BLEs into CLB resources while minimising the number of CLBs. Secondly, external CLB interconnects must be reduced by including as many connections within the CLBs. Fig. 1(c) illustrates this and explains how fewer external CLB interconnects facilitate routing [6, 15].

Under the assumption that I is the CLB input number, N specifies the BLE number within the CLB and each has one clock, the circuit clustering problem can be formulated as a set of BLEs: $B = \{b_1, b_2, ..., b_n\}$ representing a synthesized circuit, and a set of empty CLBs representing the FPGA: $C = \{c_1, c_2, ..., c_m\}$. When clustering BLEs into CLBs, the following conditions have to be met:

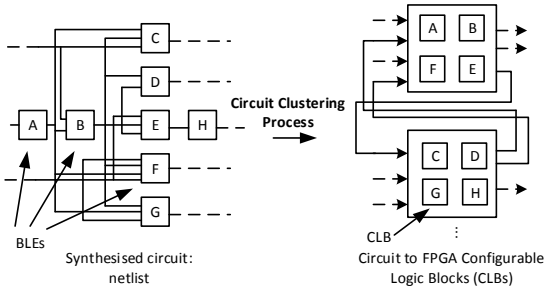
$$\text{INPUT}(c_i) \leq I \quad i = 1, 2, ..., m \tag{1}$$

$$\text{BLE}(c_i) \leq N \quad i = 1, 2, ..., m \tag{2}$$

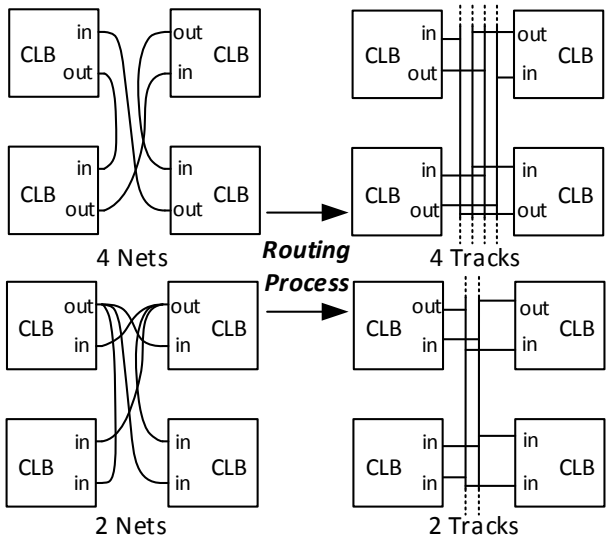
$$c_i \cap c_j = \emptyset \quad i, j = 1, 2, ..., m,\ i \neq j \tag{3}$$



(a) A typical FPGA CAD flow, including details of synthesis to logic block clustering [2]. A design is synthesised, assigned to BLEs and then clustered into CLBs. Separate placement and routing steps map the CLBs onto the FPGA fabric and connect them together.



(b) An example of the circuit clustering process [15]. Synthesized gates are packed into BLEs and then clustered in CLBs.



(c) Mapping results under different CLB interconnects, when a clustered circuit has fewer CLB interconnects, the routed circuit can have fewer tracks [15]. Therefore, a narrow channel width is used on the FPGA.

Fig. 1

$$\sum_{i=1}^{m} \text{BLE}(c_i) = B \quad i = 1, 2, ..., m \tag{4}$$

Hence, a routability-driven circuit clustering method usually optimises two aspects of a clustered circuit, which are defined in Equations (5)-(6). Equation (5) represents a circuit's absolute area

on a FPGA. As shown in Fig. 1(c), the CLB interconnect (net) number also has to be as small as possible; this condition is represented by Equation (6):

$$\text{BLE}(c_i) \rightarrow N \ (minimise \ |C|) \quad i = 1, 2, ..., m \tag{5}$$

$$\cup_{i=1}^{m} Net(c_i) \rightarrow 0 \quad i = 1, 2, ..., m \tag{6}$$

As a result, these two parameters are usually considered a "golden rule" to evaluate the quality of the clustered circuit. In addition to improving routability, the circuit clustering method also reduces the delay of the mapped circuit thereby improving its speed through optimising the critical path. Other goals include power-driven circuit clustering, however, we focus on routability- and timing-driven circuit clustering in this paper.

3 Conventional Circuit Clustering Methods

This section reviews a number of well-known circuit clustering methods (algorithms). Most methods are targeting CLB-input-bandwidth-constraint island-style FPGAs. In contrast, there are also methods for the input-bandwidth-free CLB FPGA, where I is set to $k \times N$ and represented no input constraint.

These clustering techniques can be classified as bottom-up and top-down. Bottom-up refers to clustering a circuit by moving BLEs into CLBs sequentially based on a greedy algorithm, giving a locally optimal perspective, and CLBs are constructed one by one. Top-down methods typically consider using graph partitioning methods, which view a circuit from a more global perspective, and separate the circuit by recursively partitioning it until each part of the circuit is able to fit into CLBs. There are also hybrid methods and post-routing-assisted methods. "Hybrid" methods combine bottom-up and top-down methods, and "post-routing-assistant" indicates that the methods incorporate the CAD flow in their techniques.

3.1 Bottom-up Methods

In bottom-up methods, a seed BLE has to be selected via a specified method, for example the number of BLE inputs and outputs. The seed is then directly moved into an empty CLB. To cluster more suitable BLEs in the CLB, these algorithms usually use an attraction function to determine which is the best candidate BLE that can be moved next. The attraction function is weighted by a number of clustering objectives. The value of the function is known as the "gain". The highest gain BLE is selected for each clustering iteration.

Typical bottom-up clustering methods include VPack [1] T-VPack [5], RPack [6], T-RPack [18], iRAC [3] and MO-Pack [8]. As the seed BLE selection and attraction function are required in these methods, where it is uncertain whether or not the above two functions can supply the best BLE, the solution is usually local-optimal only. Weighting objectives in a single attraction function is able to meet the multi-objective optimisation needs, but the simple weighting can also destroy the proportionality between objectives.

3.2 Top-down Methods

To facilitate top-down circuit clustering, most methods in this category are based on graph partitioning approaches treating the circuit as a hypergraph. The top-down circuit clustering methods usually deploy the hMETIS [19] hypergraph partitioning algorithm, and these methods can be viewed as various extensions of hMETIS, where hMETIS is a standalone tool for performing a k-way graph partitioning based on a multi-level paradigm.

The first notable top-down FPGA circuit clustering method was introduced in [20], and designed for input-bandwidth-constraint CLB FPGAs. This method initially uses hMETIS to partitioning a circuit coarsely, involving a second iRAC-method-based step to further optimise clustered sub circuits in order to fit into CLBs. Since there is an extra step, it degrades the quality of the hMETIS results. A more recent top-down method has been proposed, PPack (also T-PPack, short for timing-driven PPack) [13], with promising

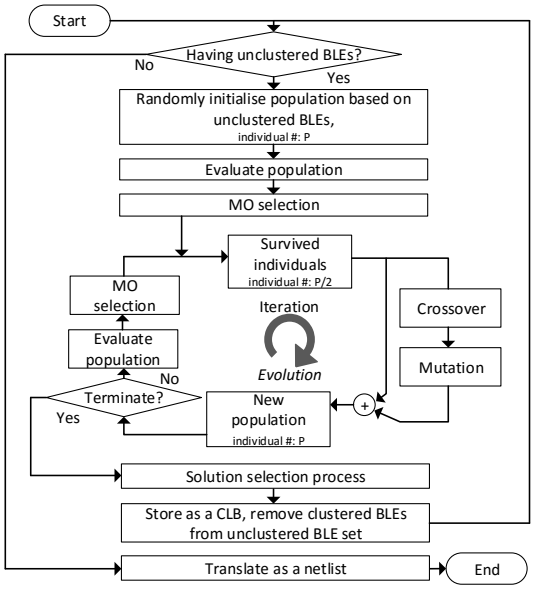


Fig. 2: DBPack circuit clustering flow. DBPack uses a number of GAs to construct CLBs. Each GA produces a set of solutions, the best solution is then used for a CLB. The flow is stopped when no unclustered BLEs are left.

results. Although these methods can produce better solutions than the bottom-up method, using graphs to cluster a circuit can make it difficult to involve clustering constraints, or clustering metrics.

3.3 Other Methods

HDPack and DPack [7] are typical examples of hybrid methods, where they use circuit partitioning to preferentially cluster the synthesised circuit into sub circuits. These sub circuits are then optimised using bottom-up methods. Moreover, HDPack also incorporates the placement process in the CAD flow (DPack without using a CAD flow), which can approximately determine which regions are more congested based on a FPGA model, and extra adjustments can be conducted for the clustered circuit. Un/DoPack [21] and T-NDPack [22] involve the entire mapping process. In addition, Un/DoPack and T-NDPack introduce the concept of depopulation [21] in their methods. Even though these methods combine both top-down and bottom-up and use the top-down method as the first step, the quality of the results is usually decreased massively during the (second) bottom-up step.

4 Proposing new MOGA-based FPGA Circuit Clustering Methods

The concept of evolutionary computing (EC) has been developed over many years [23, 24], genetic algorithms (GAs) [25] are an important variant of evolutionary algorithms developed within the EC community. The outstanding advantages of GAs are that they can produce excellent solutions for a targeted problem without significant amounts of domain knowledge introduced at the start of the process and, if designed correctly, do not fall into local minima. In addition, multi-objective genetic algorithms (MOGAs) further enhance the problem solving ability for conflicting multi-objective problems, allowing MOGAs to address real search and optimisation problems.

In this paper, we propose a novel multi-objective genetic algorithm [4, 26, 27] based circuit clustering method. The proposed method contains two customised MOGAs: DBPack and HYPack, where both use Pareto optimality to incorporate multiple optimisation objectives. The former is used to generate initial circuit clustering solutions. As the GA produces stochastic results, DBPack can

be executed many times to accumulate different solutions. HYPack then uses these solutions as input and performs further optimisation. This work is conducted from a global perspective (top-down). Subsequently, HYPack is connected to a CAD flow, and optimises solutions via the real mappings.

5 DBPack: Producing Initial Clustering Solutions Using MOGA

Rather than incrementally adding BLEs to a CLB, DBPack ("DB" being short for database), utilises a MOGA to search for groups of BLEs for a CLB. In this work, each GA run produces a CLB containing one or more BLEs. DBPack and partial experimental results were initially published in [28]. In this section, the DBPack implementation is introduced. It includes the GA chromosome representation, genetic operations, fitness functions, solution selection and experimental results.

5.1 Overview Algorithm

The DBPack execution flow is illustrated in Fig. 2. DBPack clusters BLEs by using a number of GAs, the number of GAs are dependent on whether it has un-clustered all BLEs or not. Experiments show that clustering circuits using such an approach can reduce the GA search space compared to searching for a solution from a global perspective, where a useful solution can be produced efficiently. In each GA run, the initial population is randomly generated and based on the un-clustered BLEs. Then the individuals are evolved under multiple clustering objectives.

5.2 Representation

A binary string has been used to encode DBPack GA's chromosome. The DBPack GA chromosome consists of a number of genes, and these genes are used to represent BLE selected for a CLB. The number of genes in the chromosome is determined by the un-clustered BLE number, hence the chromosome length is variable (as un-clustered BLEs become clustered). Each gene is used to encode each BLE index. A gene that has the binary value "1" means that the BLE index corresponding to that gene position has been selected for a CLB. Otherwise, the gene has the binary value "0" suggesting that the corresponding BLE is not yet selected. The detailed GA representation can be found in [28].

5.3 Reproduction

Each GA has both crossover and mutation implemented as genetic operation in order to generate new individuals. The crossover is a one-point binary crossover, and the crossover operation is controlled by a crossover rate. In a GA generation, two individuals are randomly selected from the population to perform crossover, the crossover point of their chromosomes is randomly determined. These two individuals then produce two new individuals.

The "flipping a bit" mutation operation is utilised in DBPack. This mutation operates after crossover, and is performed on all offspring. For each offspring, DBPack mutation operation is designed to randomly flip one, and only one gene in the individual's chromosome.

5.4 Fitness Functions

At each CLB construction, DBPack involves five fitness functions to guide the GA evolution to search for suitable BLEs. Each objective requires its own fitness function which evaluates a candidate solution and assigns a quality metric in the form of a fitness value to guide evolutionary search. In this case, these fitness functions not only describe which objectives need to be optimised, but also handle clustering constraints. In DBPack, the MO selection is based on the NSGA-II [4], which selects the best individuals using the fast-non-dominated sort and crowding distance.

$$f_{\mathrm{BLE}}(x) = (\#\ of\ BLEs)^{-1} \tag{7}$$

$$f_{\mathrm{inter.\ cons.}}(x) = \begin{cases} 2,\ (\#\ of\ inter.\ cons. = 0) \\ \\ (\#\ of\ inter.\ cons.)^{-1} \\ ,\ (\#\ of\ inter.\ cons. > 0) \end{cases} \tag{8}$$

$$f_{\mathrm{incresd.\ cons.}}(x) = \#\ of\ increased\ CLB\ nets \tag{9}$$

$$f_{\mathrm{input}}(x) = \#\ of\ inputs \tag{10}$$

$$f_{\mathrm{output}}(x) = \#\ of\ outputs \tag{11}$$

Clustering objectives are described in Equations (7)-(11). Each function represents one objective of the searched BLEs, and all functions are defined to return smaller values when the function represented objective is improved. Explanations of these functions are as follows: Equation (7) represents the number of BLEs for a CLB. Equation (8) shows how many circuit connections a CLB contains. It presents two situations: When the BLEs have no included connection, it returns a large penalty. Otherwise, it presents a function relationship of CLB included connections. Equation (9) is to set up a global optimisation for CLB interconnects. If there are already clustered CLBs, current clustered CLB interconnects are known. When a new CLB is added, how many new interconnects appeared is calculable. Equations (10)-(11) are the controls of the CLB input and output numbers, and these are inspired by Rent's rule [29].

$$f_{\mathrm{BLE\ penalty}}(x) = \begin{cases} 0,\ (\#\ of\ BLEs \leq N) \\ \\ \#\ of\ BLEs\ /\ A \\ ,\ (\#\ of\ BLEs > N) \end{cases} \tag{12}$$

$$f_{\mathrm{input\ penalty}}(x) = \begin{cases} 0,\ (\#\ of\ inputs \leq I) \\ \\ \#\ of\ inputs * B \\ ,\ (\#\ of\ BLEs > I) \end{cases} \tag{13}$$

In addition to the objective functions that are defined, penalties are implemented to handle constraints. Equations (12)-(13) are the defined penalty functions. These functions produce penalty violations when BLE combinations are invalid for the targeted CLB type. Equation (12) presents the BLE number constraint, and Equation (13) is to control the input number of the BLEs. A and B are two proportional coefficients. These coefficients adjust the penalty violation levels. Experiments show, $A = 7$, $B = 2$ are efficient settings, where the penalty violations have to be small enough to avoid degrading the GA population diversity.

The objective functions and penalty functions have been set up for DBPack. According to [27, 30, 31], the penalty can be added to all objective functions to handle constraints in MOGAs. DBPack fitness functions are defined as Equations (14), where Equation (15) shows the sum of the penalties. Index j indicates the five objects listed in Equations (12)-(13).

$$f_{\mathrm{fitness\ j}}(x) = f_{\mathrm{obj\ j}}(x) + f_{\mathrm{penalty}}(x) \tag{14}$$

$$f_{\mathrm{penalty}}(x) = f_{\mathrm{BLE\ penalty}}(x) + f_{\mathrm{input\ penalty}}(x) \tag{15}$$

5.5 Solution Selection

In DBPack, each CLB construction uses a MOGA, and the GA executes for a fixed number of generations. At the end of this execution all individuals can be considered as possible solutions for a CLB. In order to identify the best individual based upon MO characteristic, the selection process checks all final generation individuals.

Table 1 Results comparison between DBPack, VPack, T-VPack, RPack and iRAC for a subset of MCNC benchmarks. "Interc." represents the number of CLB interconnects, lower values are better

	Circuit Property		DBPack		VPack		T-VPack		RPack		iRAC	
Benchmark	LUTs	FFs	CLBs	Interc.	CLBs	Interc.	CLBs	Interc.	CLBs	Interc.	CLBs	Interc.
alu4	1522	0	192	543	198	1296	192	804	196	985	196	624
apex2	1878	0	238	841	241	1626	240	1249	242	1288	249	993
apex4	1262	0	161	637	163	1037	165	863	167	868	168	739
bigkey	1707	224	214	716	214	1622	214	1040	214	1060	227	585
clma	8381	33	1058	3636	1056	7139	1054	5307	1054	5585	1089	3884
des	1591	0	199	909	204	1601	200	1214	202	1339	352	1213
diffeq	1494	377	188	590	188	1280	189	1033	189	895	195	662
dsip	1370	224	172	713	198	1590	172	762	188	1219	228	472
elliptic	3602	1122	453	1278	453	3244	454	2247	462	2300	475	1408
ex1010	1598	0	587	2264	595	3799	599	3110	602	3064	613	2575
ex5p	1064	0	136	595	136	950	139	767	139	754	140	664
frisc	3539	886	447	1266	448	2955	446	2048	447	1983	477	1521
misex3	1397	0	177	586	178	1101	178	840	178	876	191	679
pdc	4575	0	580	1934	593	3813	582	2627	590	3011	608	2246
s298	1930	8	242	480	246	1711	243	767	243	1330	251	591
s38417	6096	1463	804	2912	803	4921	802	4423	802	3921	825	3153
s38584	6281	1260	806	2537	806	4649	806	4183	806	3556	839	2884
seq	1750	0	222	753	223	1496	221	1055	223	1166	223	878
spla	3690	0	467	1464	476	3031	469	2099	473	2336	484	1771
tseng	1046	385	132	464	132	979	133	801	133	764	141	535
Sum of CLBs			7475		7551		7498		7550		7971	
DBPack # CLB improvement			0.00%		1.01%		0.31%		0.99%		6.22%	
Sum of Interc.				25118		49840		37239		38300		28077
DBPack Interc. improvement				0.00%		49.60%		32.55%		34.42%		10.54%

Individuals that are on the first Pareto front and having n ($n = N$) BLEs and less than or equal I inputs, are temporarily stored. In practice, the GA might not find any solution which has $n = N$ BLEs, so n is reduced until individuals are found. The key to this process is to find all maximum BLE solutions. Subsequently, these temporarily stored individuals are ranked based on their internal connections. The individual that has the most internal connections is selected as a CLB.

5.6 Results Comparison

The largest benchmark "clma" in MCNC-20 is used for adjusting the DBPack GA parameters as it represents the largest search space. The calibrated DBPack GA parameters are summarised as follows:

1. Population size: 200
2. Crossover probability: 0.6
3. Mutation probability: one gene per individual
4. Maximum generation number: 15000

As the output of DBPack is fully clustered circuits, which enables the comparison of the clustered circuits to other methods. Similar to these other methods, the CLB size N, CLB input number I and LUT size k are set to 8, 18 and 4, and include one clock. Table 1 lists the DBPack best results compared to previously published bests, including CLB number and CLB interconnects. These results, each benchmark, are based on 100 DBPack runs. As can be seen, DBPack CLB interconnect can be reduced by up to 49.60% compared to VPack, and also better than other methods. At meantime, DBPack can maximum CLB utilisation.

6 HYPack: Optimising Clustering Solutions from A Global Perspective Using MOGA

HYPack ("HY" short for hybrid) attempts to continuously optimise clustered circuits from a global perspective, and also incorporates with DBPack a method to re-cluster BLEs. The HYPack implementation is introduced in this section including the GA chromosome representation and genetic operations. The HYPack experimental results can be found in [28], Subsequently, HYPack has been extended as a timing-driven circuit clustering method, T-HYPack, where CAD flow based fitness is involved. The fitness functions and further experimental results of T-HYPack are detailed in the next section.

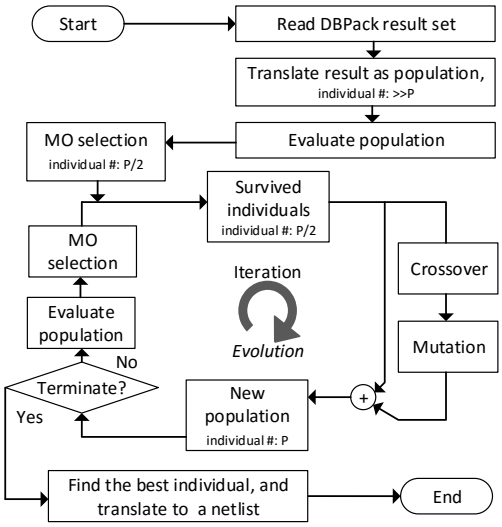
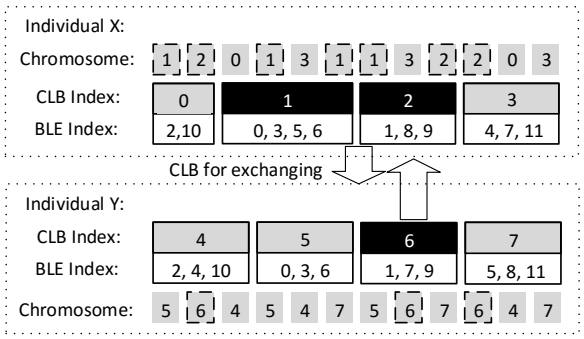


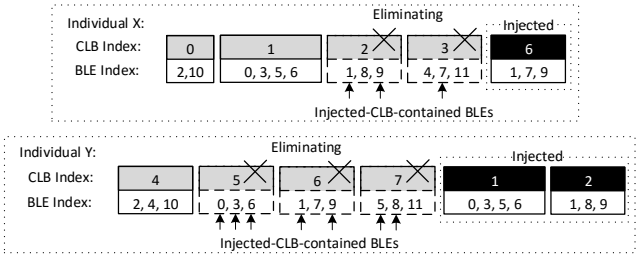
Fig. 3: HYPack circuit clustering flow. HYPack reads already clustered circuit solutions, and uses these as the GA's initial population. Then the GA further optimises solutions under an MO selection mechanism.

6.1 Overview Algorithm

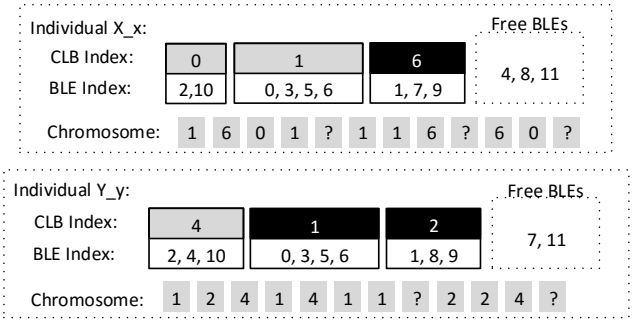
The HYPack execution flow is illustrated in Fig. 3. HYPack optimises intermediate solutions, in this case, from DBPack. Assuming DBPack has executed many times and generated enough solutions, these solutions are then converted as HYPack initial population. Subsequently, the solutions are selected via a MO selection mechanism which is similar to DBPack, and best solutions are used in the HYPack GA loop. The HYPack GA is executed for a fixed number of generations. On the final generation, an individual is chosen from the first Pareto front based on solution CLB interconnect and timing.



(a) Select the crossover CLBs from copied individuals



(b) Directly inject selected CLBs in individuals, and eliminate CLBs that contain these injected BLEs



(c) Keep freed BLEs, and store two individuals as offspring

Fig. 4: HYPack crossover operation. This operation is designed to swap BLE combinations between two possible solutions (individuals).

6.2 Representation

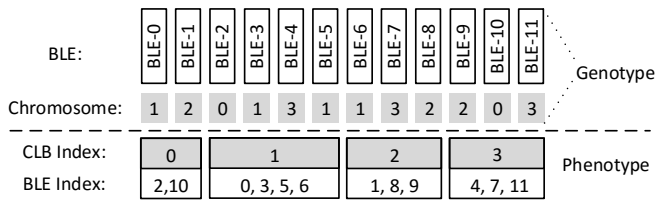
In the HYPack, an integer string has been selected to encode the chromosome. These integer values present the CLB index, and the gene's position is used to encode the BLE index. Fig. 5(a) is an example to illustrate this representation. In this representation, each integer position, gene position, is used to encode each independent BLE, and these gene values indicate which CLBs the BLEs are allocated. Inside the chromosome, its gene number is equal to BLE number, hence the length of chromosome is variable and dependent on the BLE number.

6.3 Reproduction

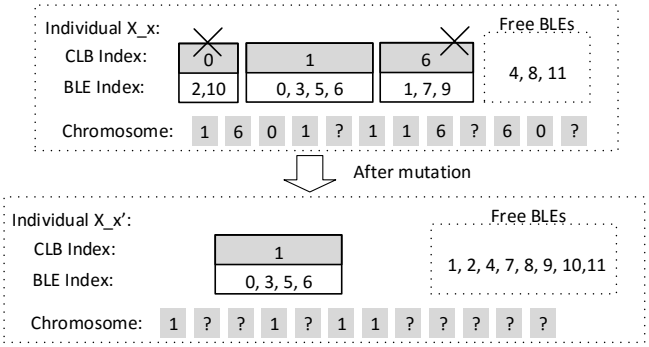
Both crossover and mutation are implemented in the HYPack GA to create new individuals, and these operations are inspired by [32]. These genetic operations have two functions: Firstly, The crossover operation is intended to exchange BLE combinations (CLBs) between different solutions. Secondly, The mutation operation is designed to generate new BLE combinations for CLBs.

HYPack GA crossover operates in five steps:

1. Select two individuals randomly from the population, and copy them to produce two new individuals.
2. Randomly determine which CLBs between the copied individuals exchange BLEs (crossover).



(a) HYPack chromosome representation. HYPack uses an integer string to represent a circuit clustering solution. The integer values present the CLB index, each BLE is also encoded by an integer indicating which CLB it belongs to.



(b) HYPack mutation operation. It randomly eliminate two CLBs in an individual, and the released CLBs are kept in the individual.

Fig. 5: HYPack genotype and mutation operation.

3. Inject the chosen CLBs into copied individuals of each individual.
4. Eliminate CLBs that contain these injected BLEs.
5. Store the two copied and crossed individuals as offspring.

Fig. 4(a) presents an example of two randomly selected and copied individuals (X and Y) from the population. In these individuals, the crossover CLBs, or the exchange-purposed CLBs, are randomly selected. For example, the crossover CLBs are the three CLBs shown in black. The number of crossover CLBs in each individual is configurable, and controlled by a pre-defined range. After determining the CLBs, these CLBs are directly injected into two individuals, as shown in Fig. 4(b). Subsequently, the crossover operation checks injected-CLB contained BLEs, and eliminates the individual CLBs that contain injected BLEs. The BLEs that do not appear in the injected CLBs need to be released for subsequent reinclusion in another CLB. Once the injection (crossover) process is finished, the injected CLBs are reserved in the injected individuals X_x and Y_y, as shown in Fig. 4(c).

The mutation operation in HYPack is executed after the crossover operation, it is designed to randomly eliminate two CLBs in one individual, which releases previously allocated BLEs in the individual. Fig. 5(b) illustrates the HYPack mutation operation. Individual X_x is an individual before the mutation. Individual X_x' is a possible individual after the mutation.

After both crossover and mutation operations are completed, a number of BLEs are likely to have been released. These released BLEs which are reserved in the individual need to be reclustered, and this is using the DBPack method. In this reclustered process, the DBPack method only produces new CLBs with new indexes back to the HYPack GA individuals. This implies that the DBPack method does not reinsert a single BLE into an individual exsited CLBs, also there is no need to handling clustering constrains in HYPack as there is no invaild solution generated.

Table 2 FPGA model details for evaluating clustered circuits in VPR. (In./Out.: input/output, conn.: connection. flex.: flexibility)

Parameter	Setting
BLE, CLB – $k, I, N, Clock$	$k, I, N = 4, 18, 8$ and one clock
Switch block type	Wilton
Switch block flexibility – F_s	3
In. conn. block flex. – $F_{c,in}$	0.25
Out. conn. block flex. – $F_{c,out}$	1
Wire segments	Segment length 4, uniform
Technology	TSMC's $0.35\mu M$ CMOS process

7 T-HYPack: Involving CAD Flow in GA for further Improving Clustering Solutions

The major difference from HYPack is that T-HYPack uses VPR [14] to facilitate a CAD-assistant, placement and routing processes, circuit timing optimisation, which extracts additional FPGA mapping information as the fitness in HYPack GA loop. This timing optimisation is different from conventional methods, for example T-VPack, where clusters target circuits based on a circuit timing analysis and incorporating the connection criticality, and then arranges most critical connections inside CLBs. The reason is that the FPGA CLB internal connection propagation delays are lower than CLB interconnects.

7.1 Extracting Mapping Parameters as Fitness Functions

When T-HYPack evolves solutions, or receives initial solutions from DBPack, the solutions are not only evaluated on the basic circuit clustering requirements, but also comparing their mapping performances.

In the solution evaluation process, an individual is firstly converted as a clustered circuit. The basic clustering fitness then are assigned, as shown in Equation (16)-(18), where (16) indicates CLB number, (17) shows the CLB interconnect number and (18) illustrates how many connections are included in CLBs. For each individual, the individual represented circuit, is then translated as a VPR readable netlist. Subsequently, the VPR output is used to compose new fitness criteria. The new fitness functions are represented in Equations (19)-(20). T-HYPack has five fitness functions, all these fitness functions return smaller values when a better solution is found.

$$f_{\mathrm{hyobj1}}(x) = \#\ of\ CLBs \tag{16}$$

$$f_{\mathrm{hyobj2}}(x) = \#\ of\ global\ nets \tag{17}$$

$$f_{\mathrm{hyobj3}}(x) = (\#\ of\ CLB\ absorbed\ nets)^{-1} \tag{18}$$

$$f_{\mathrm{hyobj4}}(x) = Critical\ path\ delay \tag{19}$$

$$f_{\mathrm{hyobj5}}(x) = Routing\ wire\ length \tag{20}$$

T-HYPack is executed for a fixed number of generations. On the final generation, the best individual, with the smallest delay and fewest CLB solution, is chosen from the first Pareto front, the individual is then converted as a netlist, and regarded as the ultimate clustering solution of a targeted circuit.

7.2 Results Comparison

According to DBPack GA parameter calibration and settings, the GA parameters of T-HYPack were determined. As the released BLEs are limited in number, generation and population size reductions are applied to the BLE recluster process. Recluster DBPack GA, T-HYPack and recluster DBPack GA parameters are summarised as follows:

1. Initial solution size: 100 (initial solution number)

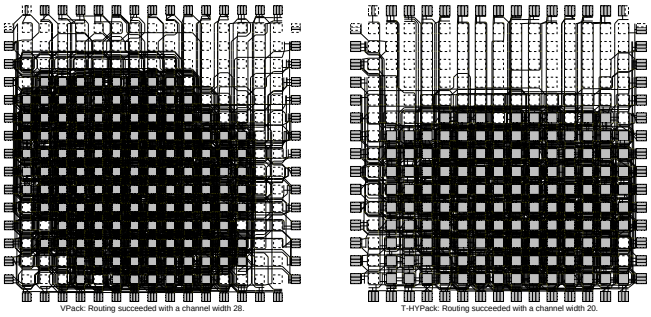


Fig. 6: Different routings of "tseng" benchmark when using VPack and T-HYPack circuit clustering methods. VPack (left) uses 28 tracks vs. T-HYPack (right) uses 20 tracks in the FPGA routing channel.

2. T-HYPack population size: 10 (keep best 10 from 100)
3. T-HYPack crossover rate: 0.6
4. T-HYPack generation number: 1,000
5. Reinsert DBPack population size: 200
6. Reinsert DBPack crossover rate: 0.6
7. Reinsert DBPack mutation rate: one gene per offspring.
8. Reinsert DBPack generation number: 2,000

As T-HYPack involves VPR, and VPR mapping is time consuming when circuits are large, which significantly increases the evolution time, T-HYPack only uses ten small MCNC-20 benchmarks for testing; "small" refers to a synthesised benchmark that has 1,000-1,500 BLEs. Based on the FPGA model, the defined FPGA architecture parameters are shown in Table 2, which are based on the T-VPack testing environments.

Each selected MCNC-20 benchmark has been optimised ten times using T-HYPack, the best results are compared to T-VPack. Table 3 shows the comparison, which includes circuit CLB number, CLB interconnect, channel width, routing wire length and delay (timing). Note that T-HYPack uses the same FPGA area as T-VPack. As can be seen, T-HYPack solution demonstrated a number of improvements, T-HYPack can speed up a circuit by up to 27.62% compared to T-VPack. As nearly all methods are compared with T-VPack, Table 4 shows a general comparison to these methods based on the literature. The table indicates the T-HYPack is the best timing-driven circuit clustering method. In addition, Fig. 6 shows the routed "tseng" benchmark on the FPGA using two different clustering methods – VPack and T-HYPack. However, T-HYPack has its disadvantages, the important concern is that the execution time as it is based on the MOGA and involving placement and routing in each GA generation, which can be slower than some methods. This issue can be aliviated somewhat by the linear speed-up that can be obtained by GAs when executed on multiple processing cores.

8 Conclusion

In this paper, we present a new method to perform circuit clustering based on MOGAs for cluster-based FPGAs. Directly forming CLBs from BLEs makes the clustering efficient and robust, and the use of Pareto optimality also allows multiple objectives to be evolved simultaneously without degrading the solution quality on a particular objective. The CAD-assisted method also shows significant potential to improve the circuit clustering. The experimental results indicate that our method offers a significant improvement on clustered circuits compared to other methods, especially for circuit timing performance. However, there is also a cost; The main concern is currently execution time which is longer than conventional clustering algorithms as the proposed method uses a combination of GA, nested GAs, and CAD flow. However the structure of our method and algorithms is inherently parallel and can be improved to reduce execution time by using parallel execution on a multi-core system or HPC infrastructure. A larger number of GA generations

Table 3 Best timing performed T-HYPack results compared to T-VPack. (Algo.:algorithm, Interc.: CLB interconnects, CH: channel width, Wire-Len: wire length, T-V: T-VPack, T-HY: T-HYPack. Lower is better)

Benchmark	Algo.	CLBs	Interc.	CH	Wire-Len.	Delay (nS)
alu4	T-V.	192	804	34	9410	8.33
	T-HY.	192	528	34	8864	7.15
apex2	T-V.	240	1249	44	15681	11.05
	T-HY.	238	834	44	15824	8.75
apex4	T-V.	165	863	52	12072	9.74
	T-HY.	162	645	46	10588	7.60
bigkey	T-V.	214	1040	26	15619	6.44
	T-HY.	214	669	14	13640	4.49
diffeq	T-V.	189	1033	28	7686	6.22
	T-HY.	188	565	26	6817	7.14
dsip	T-V.	172	762	18	14368	6.21
	T-HY.	172	704	22	15157	4.58
ex5p	T-V.	139	767	46	9780	10.01
	T-HY.	136	592	46	9618	7.68
misex3	T-V.	178	840	38	10429	8.93
	T-HY.	178	579	42	9925	7.08
seq	T-V.	221	1055	42	14480	8.93
	T-HY.	225	758	44	13800	7.22
tseng	T-V.	133	801	24	6632	7.80
	T-HY.	132	456	20	4545	6.15

Table 4 A comparison for FPGA circuit clustering methods. (CLB input-bandwidth-free circuit clustering method, N.A.: not available, Interc.: CLB interconnects. CH: Channel, Higher is better)

Method	CLB(%)	CLB interc.(%)	CH width(%)	Wire length(%)	Delay(%)
T-VPack	0.00	0.00	0.00	0.00	0.00
T-RPack	N.A.	7.01	2.66	N.A.	5.00
iRAC	-6.24	25.86	16.10	25.00	-4.35
DPack	N.A.	9.20	N.A.	17.70	7.80
HDPack	N.A.	12.70	N.A.	23.20	6.10
MO-Pack	N.A.	10.73	11.44	12.60	-1.44
PPack*	N.A.	N.A.	19.80	17.20	-4.30
T-PPack*	N.A.	N.A.	17.00	15.10	3.60
DBPack	0.59	31.21	13.00	8.70	20.85
T-HYPack	0.54	32.34	6.25	8.52	27.62

and benchmarks can then be tested, where the method can be further evaluated. The key contribution of the proposed methodology is a flexible design automation method that can incorporate multiple objectives and constraints required to successfully tackle real-world circuit design on FPGA, resulting in better FPGA device utilisation at increased circuit performance. Additional clustering objectives can be incorporated without the need to change the core algorithm.

Acknowledgment

This work is part of the PAnDA project that is funded by EPSRC (EP/I005838/1).

9 References

1 Betz, V., Rose, J. 'Cluster-Based Logic Blocks for FPGAs: Area-Efficiency vs. Input Sharing and Size'. In: CICC. (, 1997. pp. 551–554

2 Betz, V., Rose, J., Marquardt, A.R.: 'Architecture and CAD for Deep-Submicron FPGAs'. Betz, V., Rose, J., Marquardt, A., editors. (Norwell, MA, USA: Kluwer Academic Publishers, 1999)

3 Singh, A., Marek.Sadowska, M. 'Efficient Circuit Clustering for Area and Power Reduction in FPGAs'. In: FPGA '02 Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays. (ACM, 2002. pp. 59–66

4 Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: 'A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II', *Evolutionary Computation, IEEE Transactions*, 2002, **6**, pp. 182–197

5 Marquardt, A.S., Betz, V., Rose, J. 'Using Cluster-based Logic Blocks and Timing-driven Packing to Improve FPGA Speed and Density'. In: FPGA '99 Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays. (ACM, 1999. pp. 37–46

6 Bozorgzadeh, E., Ogrenci.Memik, S., Sarrafzadeh, M. 'RPack: Routability-driven Packing for Cluster-based FPGAs'. In: Design Automation Conference, 2001. Proceedings of the ASP-DAC 2001. Asia and South Pacific. (IEEE, 2001. pp. 629–634

7 Chen, D.T., Vorwerk, K., Kennings, A. 'Improving Timing-driven FPGA Packing with Physical Information'. In: Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on. (IEEE, 2007. pp. 117–123

8 Rajavel, S.T., Akoglu, A. 'MO-Pack: Many-objective Clustering for FPGA CAD'. In: Proceedings of the 48th Design Automation Conference. (ACM, 2011. pp. 818–823

9 Kuon, I., Tessier, R., Rose, J.: 'FPGA Architecture: Survey and Challenges', *Foundations and Trends in Electronic Design Automation*, 2007, **2**, (2), pp. 135–253

10 Feng, W., Kaptanoglu, S.: 'Designing Efficient Input Interconnect Blocks for LUT Clusters Using Counting and Entropy', *ACM Trans Reconfigurable Technol Syst*, 2008, **1**, (1), pp. 6:1–6:28. Available from: `http://doi.acm.org/10.1145/1331897.1331902`

11 Lewis, D. 'The stratix™ logic and routing architecture'. In: Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field Programmable Gate Arrays. (ACM, 2003. pp. 12–20

12 Leventis, P., Chan, M., Chan, M., Lewis, D., Nouban, B., Powell, G., et al. 'Cyclone: A Low-cost, High-performance FPGA'. In: In Proceedings of the IEEE 2003 CICC. (, 2003. pp. 49–52

13 Feng, W. 'K-way Partitioning Based Packing for FPGA Logic Blocks without Input Bandwidth Constraint'. In: Field-Programmable Technology (FPT), 2012 International Conference. (IEEE, 2012. pp. 8–15

14 Betz, V., Rose, J. 'VPR: A New Packing Placement and Routing Tool for FPGA Research'. In: Int Workshop on Int. Workshop on Field-Programmable Logic and Application. (, 1997. pp. 213–222

15 Marquardt, A.R. 'Cluster-Based Architecture, Timing-Driven Packing and Timing-Driven Placement for FPGAs'. University of Toronto, 1999

16 Rose, J., Brown, S.: 'Flexibility of Interconnection Structures for Field-Programmable Gate Arrays', *Solid-State Circuits, IEEE Journal*, 1991, **26**, (3), pp. 277–282

17 Fukunaga, A.S., Korf, R.E.: 'Bin completion algorithms for multicontainer packing, knapsack, and covering problems', *Journal of Artificial Intelligence Research*, 2007, pp. 393–429

18 Bozorgzadeh, E., Memik, S.O., Yang, X., Sarrafzadeh, M.: 'Routability-driven Packing: Metrics and Algorithms for Cluster-based FPGAs', *Journal of Circuits, Systems, and Computers*, 2004, pp. 77–100

19 Karypis, G., Kumar, V. 'Multilevel K-way Hypergraph Partitioning'. In: DAC '99 Proceedings of the 36th annual ACM/IEEE Design Automation Conference. (ACM, 1999. pp. 343–348

20 Marrakchi, Z., Mrabet, H., Mehrez, H. 'Hierarchical FPGA Clustering Based on A Multilevel Partitioning Approach to Improve Routability and Reduce Power Dissapation'. (IEEE, 2005. p. 25

21 Tom, M., Leong, D., Lemieux, G. 'Un/DoPack: Re-clustering of Large System-on-Chip Designs with Interconnect Variation for Low-cost FPGAs'. In: Computer-Aided Design, 2006. ICCAD'06. IEEE/ACM International Conference on. (IEEE, 2006. pp. 680–687

22 Liu, H., Akoglu, A. 'T-NDPack: Timing-driven Non-uniform Depopulation Based Clustering'. In: Programmable Logic, 2009. SPL. 5th Southern Conference on. (IEEE, 2009. pp. 9–14

23 Bäck, T., Hammel, U., Schwefel, H.P.: 'Evolutionary Computation: Comments on the History and Current State', *Evolutionary computation, IEEE Transactions on*, 1997, **1**, (1), pp. 3–17

24 Eiben, A.E., Smith, J.E.: 'Introduction to Evolutionary Computing'. Natural Computing Series. (Springer Berlin Heidelberg, 2007). Available from: `https://books.google.co.uk/books?id=RRKo9xVFW\_QC`

25 Holland, J.H.: 'Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence'. (U Michigan Press, 1975)

26 Horn, J., Nafploitis, N., Goldberg, D.E. 'A Niched Pareto Genetic Algorithm for Multiobjective Otimization'. In: Michalewicz, Z., editor. Proceedings of the First IEEE Conference on Evolutionary Computation,. (Piscataway NJ: IEEE Press, 1994. pp. 82–87

27 Srinivas, N., Deb, K. 'Multiobjective Function Optimization Using Nondominated Sorting Genetic Algorithms'. In: Evol. Comput.. vol. 2. (, 1995. pp. 221–248

28 Wang, Y., Bale, S.J., Walker, J.A., Trefzer, M.A., Tyrrell, A.M. 'Multiobjective genetic algorithm for routability-driven circuit clustering on fpgas'. In: 2014 IEEE International Conference on Evolvable Systems. (, 2014. pp. 109–116

29 Landman, B.S., Russo, R.L.: 'On a Pin Versus Block Relationship For Partitions of Logic Graphs', *Computers, IEEE Transactions*, 1971, **C-20**

30 Deb, K., Pratap, A., Meyarivan, T. 'Constrained Test Problems for Multi-Objective Evolutionary Optimization'. In: Evolutionary Multi-Criterion Optimization. (Springer, 2001. pp. 284–298

31 Deb, K., Pratap, A., Moitra, S. 'Mechanical Component Design for Multiple Objectives Using Elitist Non-dominated Sorting GA'. In: Proceedings of the Parallel Problem solving from Nature VI Conference. (, 2000. pp. 859–868

32 Falkenauer, E.: 'A New Representation and Operators for Genetic Algorithms Applied to Grouping Problems', *Evolutionary Computation*, 1994, **2**, (2), pp. 123–144