FPGA Placement Improvement Using a Genetic Algorithm and the Routing Algorithm as a Cost Function

Francisco Javier Veredas and Enrique J. Carmona
Department of Artificial Intelligence
Universidad Nacional de Educación a Distancia, Madrid, Spain
Email: fveredas2@alumno.uned.es and ecarmona@dia.uned.es

***Abstract*—In this paper the placement cost function in Field-Programmable Gate-Arrays (FPGAs) is investigated. It is found that the minimization of the traditional cost function does not ensure the minimization of the critical path. This opens an opportunity to investigate different cost functions. In the paper, it is shown that using the routing algorithm as a cost function improves the placement in the optimization of the final critical path. It is also found that using this new cost function, a genetic algorithm has advantages over the traditional simulated annealing method.**

I. Introduction

FPGAs [1] are configurable devices that can be used to implement any digital hardware design. FPGAs typically contain built-in hardwired processors, substantial amounts of SRAM memory blocks, clock management systems and very fast device-to-device board-level signaling technologies. FPGAs are used in a wide variety of applications like data processing, storage, instrumentation, network communications, or digital signal processing.
A typical design flow for FPGAs consists of three major stages [2]: circuit synthesis, design implementation, and FPGA bit-stream upload. Here we are only interested in the second stage. Specifically, the design implementation stage consists of four steps: packing, placement, routing, and bit-stream generation. One of the most critical steps is the placement, where the specific location of each Configurable Logic Block (CLB) and Input/Output (I/O) block on the target FPGA is determined. Figure 1 shows an FPGA architecture with a placed circuit. Finally, in this paper, we investigate how to improve the placement quality.
This paper is organized as follows. In Section II, the placement problem is presented. The alternatives of the FPGA placement are presented in Section III. The methodology used for the experiments is described in Section IV. Section V presents the experimental results and their discussion. Finally, the last section provides concluding remarks.

II. Placement Problem in FPGAs

In this paper, we are only interested in the problem of placement. Although the problem of the best placement is NP-hard [3], a placement solution (not necessarily optimal) can be found always that the number of physical logic resources of the FPGA is greater than the logic instances of the synthesized circuit.

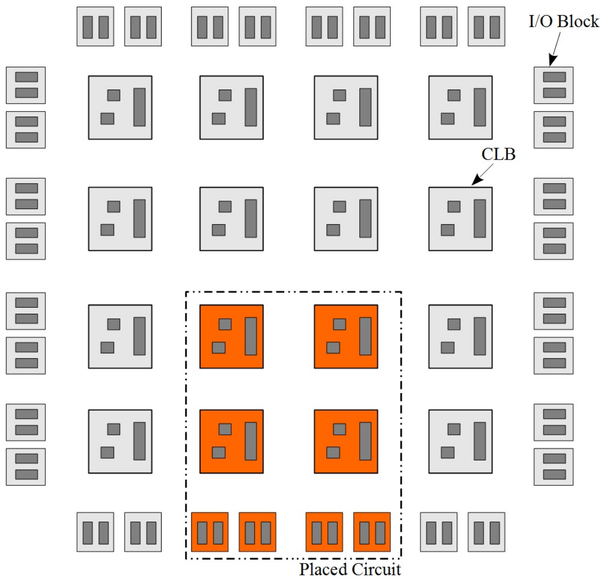


Fig. 1: FPGA Architecture showing a four CLBs and four I/O Blocks placed circuit.

One important goal for FPGA placement is to obtain a configuration of CLBs that can be successfully interconnected in a subsequent routing step. So, the evaluation of the best placement is dependent of the routing architecture, and the placement algorithm. From the designer's point of view, the problems related to the placement are: the high computational cost of the placement algorithm, a placement solution for which a routing cannot be found, and a bad quality of placement metrics (area, critical path delay, and power consumption). The placement algorithm tries to optimize the circuit performance (critical path), area (minimum array size and routing resources), and power consumption of the FPGA.
As the critical path is one of the most important optimization parameters, here we will focus on optimizing its value. The critical path is defined as the maximum delay path between all the logic paths from an input to an output.
We can find several methods to solve the placement problem of FPGAs in the literature: simulated annealing (SA) [4] [5] [6]

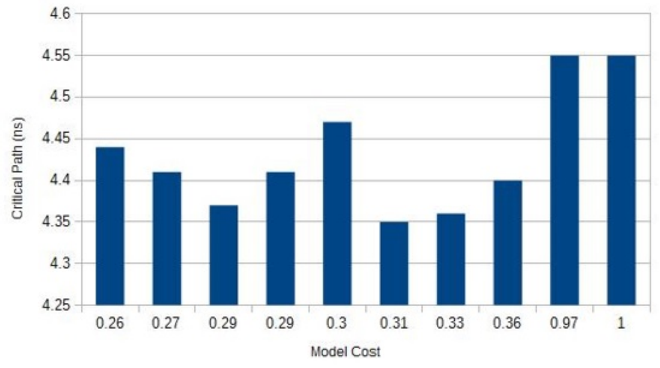
[7], genetic algorithm (GA) [8] [9] [10] [11] [12] and, analytic method [13] [14] [15] [16]. SA is the most used placement method in FPGAs and it is used for some commercial FPGAs [17]. Some studies (e.g. [12]) show no advantage in using GA versus SA. It is also found in the literature (e.g. [18] [19] [13]) that SA outperforms analytical methods in the quality metrics of FPGA placement.

On the other hand, the cost function of the placement algorithm depends on the FPGA architecture and the desired optimization. As one of the objectives of the FPGA placement is to determine the FPGA routing, the wiring congestion (how many tracks are used in one channel) is an important metric. Therefore, the final cost function should have a wire cost term [5]. Note that the configurable hardware resources to perform the routing in an FPGA are fixed. In a 2-D array FPGA (Figure 1), the tracks per channel are the configurable wires that determine the routing [1]. In most of the cases, we also want to minimize the circuit delays associated with the placement. These delays can be modeled with a timing cost term [6]. Therefore, the traditional cost function to optimize the critical path has the form:

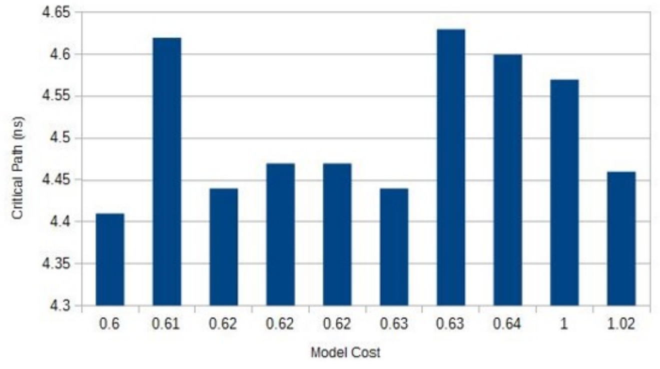
$$Cost_{wire,time} = \lambda \cdot timing\_cost + (1 - \lambda) \cdot wire\_cost \tag{1}$$

where $timing\_cost$ is a normalized time cost factor, $wire\_cost$ is a normalized wiring cost factor, and λ is a trade-off parameter. An example of using the traditional cost function is [6], where it is found that, in average, the best compromise between wire congestion and critical path cost is obtained for $\lambda = 0.5$.

The extensive use of the traditional cost function by researchers in the last twenty years allows us to think about its validity. However, an open question is if this cost function is accurate enough. We performed an experiment using the VTR placement and routing tool [20] and the circuit $s1238$ from the MCNC benchmark [21]. Figure 2 shows the critical path versus the cost function after routing is determined for circuit $s1238$. This circuit was tested with different number of iterations and using $\lambda = 0.5$ and $\lambda = 1$. To compare the results obtained from the different runs, the same normalization values were used. In particular, the normalization values are chosen with the resulting $wire\_cost$ and $timing\_cost$ of the first run. We can see in the mentioned figure that, given two different cost values, if the former is smaller than the second, the same order relation is not always guaranteed with the respective critical path values. For example, in Figure 2(a), a $cost = 0.26$ has associated in our experiment a critical path of 4.44 ns. However, in another test, a $cost = 0.33$ has associated a critical path of 4.36 ns. This indicates that if we have a placement algorithm that optimizes the traditional cost function, it will not necessarily optimize the critical path. We repeated the same experiment with other two circuits ($planet$ and $mm30a$) from the MCNC benchmark. With these circuits, we also observed the same behavior between the values of the traditional cost function and final critical path.



a) λ=0.5



b) λ=1

Fig. 2: Critical path obtained for different model cost, see eq. (1), after routing and using the VTR's tool in the $s1238$ circuit. This experiments shows that a decrease in cost values does not guarantee a decrease in the critical path.

III. New Solutions to the Placement problem

Nowadays, the increase of computation power opens the possibility of using the routing algorithm directly as a cost function. In this case, we extract the timing-driver routing algorithm of the VTR tool and insert it as a function in our placement algorithm. Every time that this function is called, the placement and its associated structures are loaded. After the routing is performed, this new cost function returns the critical path to be used as a cost value. This algorithm is called VTR with routing (VTR-R).

On the other hand, as the new cost function is a complex algorithm, it will consume more execution time than the traditional cost function. In this context, we investigate the possibility of using a GA, instead of the traditional SA, in order to try of reducing the number of cost function evaluations needed to achieve the convergence. This new algorithm is called genetic algorithm for routing (GA-R). Therefore, it will be important to analyze how many times (evaluations) the cost function is called during the execution of the placement algorithm. The final number of evaluations of the cost function in the SA algorithm is [22]:

$$Eval_{SA} = (num\_gen + 1) \cdot num\_blocks^{1.3333} \tag{2}$$

where num_gen is the number of generations, and num_blocks is the number of blocks of the circuit. The number of blocks includes the CLB blocks and I/O blocks. The number of generations depends on the exit criterion. This criterion is satisfied when $T<0.005 \cdot \frac{cost}{number\_of\_nets}$, where T is the SA temperature and $number\_of\_nets$ is the number of nets of the circuit. The temperature update is carried out by means of the expression $T_{new} = \alpha \cdot T_{old}$, where the α is a parameter that depends of the new accepted CLB placements per generation divided by the maximum number of blocks movements [5].

As we can see from the eq. 2, the number of cost function evaluations is not linear in relation to the number of blocks in the circuit. On the other hand, in a GA, the number of cost function evaluations ($Eval_{GA}$) depends on the number of generations (num_gen) and population size (pop_size), that is, it is independent of the number of blocks of a circuit:

$$Eval_{GA} = num\_gen \cdot pop\_size \tag{3}$$

Another advantage of GAs (though not explored in this paper) is that they are easily parallelizable [23] [24]. The SA algorithm used in [5] has only one thread, where a new solution depends on the previous one. On the other hand, in a GA, each individual of the population can be run independently in a thread. For example, if we have a population of 100 individuals, we could run in parallel each individual in a cluster of 100 cores. In this case, the execution time would depend only of the number of generations (according to eq. 3). Note that a circuit with a large number of blocks will also require a longer execution time, both with an SA and a GA, because the routing is also much more complex.

IV. Experiments Methodology

The FPGA architecture parameters used are shown in Table I (refer to [22] for a detailed description of these parameters). As the purpose of this paper is the study of placement, we do not investigate the routing algorithm. So, in our implementation we use a fixed channel width as it is usual in FPGA placement investigations, e.g. [25]. The other parameters are the default ones of the VTR tool and they emulate the commercial Altera Stratix IV [26]. We use the Wilton switch block type [27].

The characteristics of the circuits used can be seen in Table II. These circuits are provided by the VTR framework in BLIF format [28] and were mapped into CLBs blocks using the T-VPack tool [29].

Note that the placement can be done for CLB and I/O blocks or, alternatively, we can fix the placement for the I/O blocks (or CLB blocks) and perform the placement only for CLB blocks (or I/O blocks). In our experiments, to reduce complexity, we fix the placement of the I/O blocks. The I/O blocks placement file is obtained previously by making a placement with the SA algorithm.

To obtain the correct critical path time, we perform the routing step with the placement files coming from the SA algorithm. The initial temperature of the SA is $init\_temperature = 20 \cdot std\_dev$, where the standard deviation, std_dev, is calculated with the cost variation of moving blocks randomly. The cost of the initial random placement is obtained from the first time that the normalization cost is calculated. In the SA algorithm, with the traditional cost function, we used $\lambda = 0.5$. To find λ, we performed a set of tests with λ going from 0 to 1, and we found that $\lambda = 0.5$ was the value with best final critical path.

Algorithm 1 shows our implementation of the GA-R. This GA uses a tournament size of two individuals, a one-point crossover operator, and a mutation operator based in CLB permutations. In the GA, it is also necessary to set the number of generations, and the probabilities of mutation (Pm) and crossover (Pc). The number of generations was set to have a better number of cost function evaluations than that obtained with SA. The probabilities ($Pm = 0.04$ and $Pc = 0.5$) were tuned by means of the Local Unimodal Sampling (LUS) method [30], using the circuit $s1238$.

A workstation with two Intel E7 Xeon processors with 32GB of RAM was used for the experiments.

For each circuit, each algorithm was run 30 times. To create a different heuristic in each run, we changed the deterministic random function of the VTR tool to a semi-random function [31].

TABLE I: FPGA architecture parameters used in the experiments.

Parameter	Description	Value used
W	Channel Width	200
N	#BLE per CLB	10
K	#Inputs BLE	6
L	Segment distance of a track	4
F_{cin}	Ratio of tracks connected to an input	0.15
F_{cout}	Ratio of tracks connected to an output	0.1
F_s	Switch block type	3 (Wilton)

TABLE II: Circuit characteristics and array size used for the placement.

Circuit	LUTs	FFs	LB	I/O	Nets	Array Size
styr	238	5	15	20	105	4x4
planet	266	6	17	27	127	5x5
s1238	292	18	18	29	148	5x5
vda	253	0	19	56	176	5x5
daio-rec	311	81	19	62	230	5x5
mm30a	294	90	21	64	230	5x5
ecc	291	109	22	26	178	5x5
ex4p	148	0	22	112	206	5x5
C2670	214	0	15	221	305	7x7
rot	242	0	17	242	293	8x8
x3	255	0	20	234	281	8x8
i7	103	0	11	266	266	9x9
frg2	347	0	26	282	342	9x9

Algorithm 1 Genetic Algorithm (GA-R).

```
Input: circuit netlist, GA parameters, FPGA parameters
Output: placement netlist
 1: load circuit and FPGA structure;
 2: random placement;
 3: calculate cost;
 4: save best individual;
 5: while not termination (number generations) do
 6:   for all population do
 7:     select two random candidates from all population;
 8:     set parent1 from best of two candidates;
 9:     select two random candidates from all population;
10:     set parent2 from best of two candidates;
11:     if uniform_random_probability(0,1) <$P_c$ then
12:       cross parent1 and parent2;
13:       save new two children in new population;
14:     else
15:       copy parent1 and parent2 in new population;
16:     end if
17:   end for
18:   for all new population do
19:     for each gen in chromosome do
20:       if uniform_random_probability(0,1) <$P_m$ then
21:         mutate gen;
22:       end if
23:     end for
24:   end for
25:   resolve location conflicts;
26:   calculate total cost;
27:   if best_new_population >best_old_population then
28:     save best_new_population as best individual;
29:   else
30:     replace worst in new population with
        best_old_population;
31:   end if
32: end while
33: save best placement;
```

V. Experimental Results and Discussion

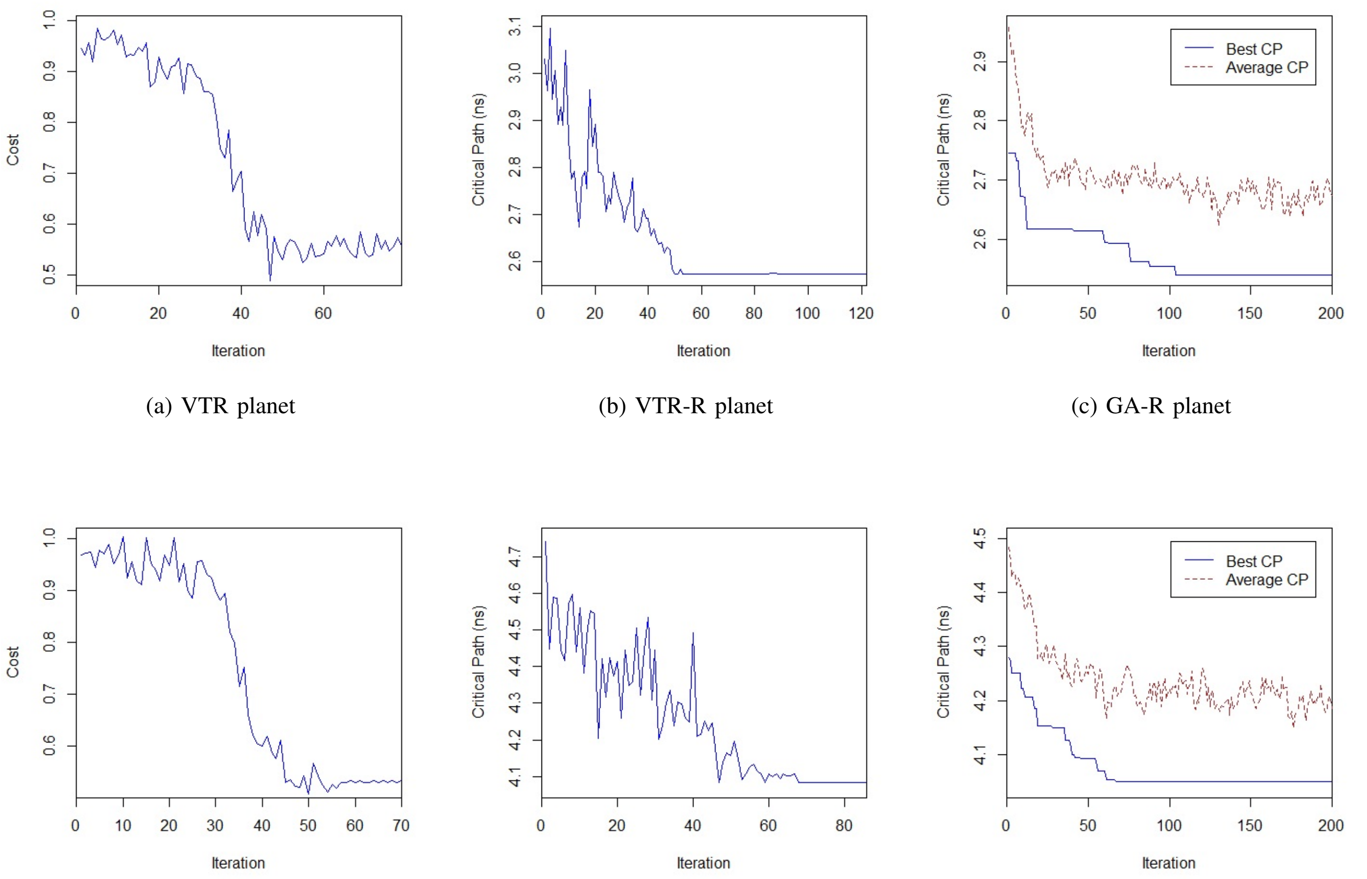
The results obtained by the algorithms VTR, VTR-R and GA-R are shown in Table III. As it can be seen in the mentioned table, the use of the new cost function (VTR-R and GA-R) improves the average critical path in comparison with the traditional cost function (VTR). This improvement is obtained at the cost of an increase of the required execution time, which is in the order of 10^5 or 10^6 higher. The critical paths between VTR-R and GA-R are similar. But a noticeable reduction in the execution time can be observed with GA-R when this is compared with VTR-R.

In order to compare the experimental results from a statistical point of view, we performed a nonparametric bootstrap hypothesis test [32]. A nonparametric test was needed because the data associated to each circuit rejected the null hypothesis of normality (Shapiro-Wilk test [33]). The significance level was set to $\alpha = 0.05$. As it can be seen from the p-values obtained in Table IV, the null hypothesis (there is no difference between the two population means) is always rejected for all the cases of VTR-R vs. VTR and GA-R vs. VTR. On the other hand, the null hypothesis cannot be rejected in all the cases of GA-R vs. VTR-R. Therefore, from the hypothesis test results, we can say that there exists statistical evidence to affirm that the critical path values obtained for VTR-R and GA-R are better than those obtained for VTR. However, we must assume similar critical path results between GA-R and VTR-R.

Figure 3 shows an example of the convergence of two circuits (planet and s1238). We can observe the typical random-walk of the SA. This happens because the temperature parameter of the SA forces the acceptance of placement solutions even with worst cost. Looking at these graphics, we can ask if there is not a possibility of stop the algorithm VTR-R before it reachs the final iteration. As it can be seen in Figure 4, there is the possibility of improving the critical path in the last iterations. In the same figure, we can also see a behavior that illustrates one of the typical problems with an SA algorithm: a good critical path is found around the iteration 45, but because it is possible to accept a placement solution worse than the current one, the placement solution deteriorates. To solve this problem, we could update and store the best solution obtained during the search process and provide it as a solution if it is better than the SA final solution.

Table V shows the cost function evaluations for the three algorithms. The VTR-R algorithm needs more evaluations than the VTR algorithm. This is because the termination criterion of the VTR is fulfilled when not better placements are found. Meanwhile the fine granularity of the cost function in VTR-R reaches improvements in new generations. We can also see that the number of evaluations in GA-R is a constant defined by eq. 3. Note that the GA-R needs fewer cost function evaluations than VTR and VTR-R to get a better or similar result.

An open question is how perform the GA whether the traditional cost function is also used. The use of GAs with the traditional cost function was investigated in [12]. In this paper, the authors found that the results obtained by a GA are similar to those obtained by an SA algorithm. These results are in line with those obtained in [12]. As the number of cost function evaluations in the SA can be higher than the GA, we can ask if it is possible to get a better execution time using the GA with the traditional cost function. However, some tests with our GA showed that the answer to that question is negative. This is because our GA was not optimized for execution time and, additionally, the traditional cost function was designed for working in an SA algorithm. For instance, the traditional cost function has a normalization that it is keep constant for several evaluations during a generation. This is not possible with the GA. Another problem, already mentioned in section II, is that the minimization of the traditional cost function does not always guarantee a minimization of the critical path (see Figure 2). This is not a big problem with the SA, where the cost function is evaluated for one single CLB movement



(a) VTR planet (b) VTR-R planet (c) GA-R planet

(d) VTR s1238 (e) VTR-R s1238 (f) GA-R s1238

Fig. 3: Convergence of the best result of the $planet$ and $s1238$ circuit. a) and d) show the evolution of the traditional cost function value using SA (VTR). b) and e) show the evolution of the critical path of the VTR with the new cost function (VTR-R). c) and f) show the evolution of the critical path average (brown) and the best critical path (blue) using the GA-R.

(an error in one cost function evaluation is compensated with the next evaluation). On the other hand, with the GA, the cost function is evaluated after several CLB movements (with crossover and mutations). So, if the best individual is found with a bad cost function prediction, it will pass to the next generation and produce a wrong offspring.

Finally, it would have been interesting to compare the results of the GA-R algorithm with other proposals that use the routing as a cost function. However, as far as the authors know, there are no such approaches in the literature. In addition, it is difficult to make a fair comparison with other related works because their objective to be optimized or the circuits used are different from those used here.

VI. Conclusion and Future Work

In this paper, we investigated the cost function of the placement algorithm. We found that the minimization of the traditional cost function used in the traditional SA algorithm does not always produce a minimal critical path. To address this problem, we proposed to use the routing algorithm as a cost function. The experimental results showed that the quality of the placement is improved using this new cost function (VTR-R and GA-R). However, the observed drawback was a longer required execution time. To reduce the execution time with the new cost function, we proposed the use of a genetic algorithm (GA-R). It was found that the GA-R improves the execution time, maintaining a competitive critical path. The new cost function will be useful in those cases where a minimum critical path is needed.

While this paper has shown the benefits of using GA and routing algorithm as a cost function, many possibilities are still open to improve the results: (1) the inherent parallelism of GAs can be exploited for improving the execution time; (2) instead of using a fixed number of generations for GA-R, a termination criterion depending on the convergence of the algorithm, can be investigated; (3) a better routing algorithm as a cost function can be investigated (instead of the default one used in this work); and (4) in order to improve the placement

TABLE III: Experimental results for critical path and execution time averaged over 30 runs. SD = Standard Deviation, Δ_{c1} and Δ_{c2} shows the critical path's difference between the VTR with VTR-R and GA-R respectively, Δ_{t1} and Δ_{t2} shows the execution time's difference between the VTR with VTR-R and GA-R respectively.

	Crit.Path(ns)					Exec.Time(sec)				
Circuit	VTR ± SD	VTR-R ± SD	Δ_{c1}%	GA-R ± SD	Δ_{c2}%	VTR ± SD	VTR-R ± SD	Δ_{t1}%	GA-R ± SD	Δ_{t2}%
styr	3.08 ± 0.16	2.71 ± 0.02	-11.71	2.72 ± 0.02	-11.62	0.37 ± 0.05	1853 ± 377	4.98E5	1365 ± 110	3.67E5
planet	2.85 ± 0.08	2.59 ± 0.02	-9.26	2.60 ± 0.02	-8.83	0.65 ± 0.05	5031 ± 1127	7.78E5	3592 ± 328	5.55E5
s1238	4.44 ± 0.09	4.11 ± 0.02	-7.46	4.11 ± 0.03	-7.45	1.71 ± 0.62	5906 ± 1510	3.43E5	3969 ± 310	2.31E5
vda	3.24 ± 0.06	2.96 ± 0.03	-8.65	2.97 ± 0.03	-8.56	1.85 ± 0.78	8977 ± 1720	4.84E5	6318 ± 968	3.41E5
daio-rec	3.97 ± 0.08	3.47 ± 0.02	-12.62	3.49 ± 0.03	-12.23	0.95 ± 0.04	11960 ± 750	1.26E6	3161 ± 134	3.34E5
mm30a	13.16 ± 0.13	12.47 ± 0.03	-5.25	12.50 ± 0.04	-5.02	0.98 ± 0.05	11802 ± 1390	1.20E6	3395 ± 121	3.46E5
ecc	3.01 ± 0.07	2.75 ± 0.03	-8.79	2.75 ± 0.03	-8.81	0.78 ± 0.03	8617 ± 2051	1.10E6	5902 ± 897	7.55E5
ex4p	2.74 ± 0.04	2.60 ± 0.02	-5.03	2.64 ± 0.03	-3.67	3.35 ± 0.85	25236 ± 7181	7.54E5	6536 ± 1713	1.95E5
C2670	3.74 ± 0.08	3.41 ± 0.03	-8.79	3.53 ± 0.07	-5.70	12.12 ± 8.74	90184 ± 20692	7.44E5	9199 ± 2173	7.58E4
rot	3.78 ± 0.03	3.57 ± 0.04	-5.86	3.59 ± 0.06	-4.77	15.42 ± 10.06	137549 ± 23390	8.92E5	11470 ± 3230	7.43E4
x3	2.56 ± 0.00	2.23 ± 0.03	-13.01	2.20 ± 0.02	-14.01	11.52 ± 3.13	170177 ± 10699	1.48E6	12311 ± 2171	3.41E5
i7	1.93 ± 0.00	1.67 ± 0.01	-13.39	1.66 ± 0.01	-13.79	19.81 ± 4.38	241126 ± 15327	1.22E6	11180 ± 2942	5.63E4
frg2	3.31 ± 0.01	3.00 ± 0.03	-9.28	3.14 ± 0.05	-4.94	14.68 ± 2.17	258359 ± 18465	1.76E6	12229 ± 4328	8.32E4

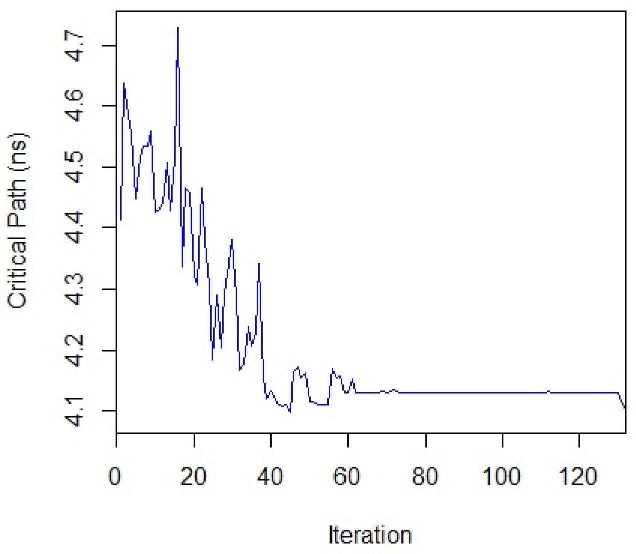


Fig. 4: Convergence of a $s1238$ circuit with SA with the new cost function (VTR-R). The main critical path optimization is done in half of the iterations, at the end there is a small improvement in the critical path.

TABLE IV: Results of the nonparametric Bootstrap hypothesis test (p-values).

Circuit	VTR-R vs. VTR	GA-R vs. VTR	GA-R vs. VTR-R
styr	0.0001	0.0001	0.6991
planet	0.0001	0.0001	0.1104
s1238	0.0001	0.0001	1
vda	0.0001	0.0001	1
daio-rec	0.0001	0.0001	0.0594
mm30a	0.0001	0.0001	0.0645
ecc	0.0001	0.0001	1
ex4p	0.0001	0.0001	1
C2670	0.0001	0.0001	1
rot	0.0001	0.0001	1
x3	0.0001	0.0001	1
i7	0.0001	0.0001	1
frg2	0.0001	0.0001	1

quality, the routing parameters (e.g. routing effort, number of tracks, etc.) can also be investigated.

References

[1] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic, *Field-Programmable Gate Arrays*. Springer Science & Business Media, 1992, vol. 180.

[2] D. Chen, J. Cong, P. Pan *et al.*, "FPGA design automation: A survey," *Foundations and Trends® in Electronic Design Automation*, vol. 1, no. 3, pp. 195–330, 2006.

[3] N. A. Sherwani, *Algorithms for VLSI physical design automation*. Springer Science & Business Media, 2012.

[4] C. Ebeling, L. McMurchie, S. A. Hauck, and S. Burns, "Placement and routing tools for the Triptych FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 3, no. 4, pp. 473–482, 1995.

[5] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," in *Field-Programmable Logic and Applications*. Springer, 1997, pp. 213–222.

[6] A. Marquardt, V. Betz, and J. Rose, "Timing-driven placement for FPGAs," in *Proceedings of the 2000 ACM/SIGDA eighth international symposium on Field programmable gate arrays*. ACM, 2000, pp. 203–213.

[7] K. Vorwerk, A. Kennings, and J. W. Greene, "Improving simulated annealing-based FPGA placement with directed moves," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 2, pp. 179–192, 2009.

[8] Z. Baruch, O. Creţ, and H. Giurgiu, "Genetic algorithm for FPGA placement," in *Proceedings of the 12th International Conference on Control Systems and Computer Science (CSCS12)*, vol. 2, 1999, pp. 121–126.

[9] R. Venkatraman and L. M. Patnaik, "An evolutionary approach to timing driven FPGA placement," in *Proceedings of the 10th Great Lakes symposium on VLSI*. ACM, 2000, pp. 81–85.

TABLE V: Number of cost function evaluations averaged over 30 runs (SD = Standard Deviation).

Circuit	Number of Evaluations.		
	VTR ± SD	VTR-R ± SD	GA-R ± SD
styr	7969 ± 434	12510 ± 2408	8400 ± 0
planet	11754 ± 447	19207 ± 2737	14000 ± 0
s1238	12603 ± 402	21470 ± 4077	14000 ± 0
vda	25250 ± 903	23680 ± 898	14000 ± 0
daio-rec	28279 ± 1154	52662 ± 815	14000 ± 0
mm30a	30024 ± 902	47319 ± 3939	14000 ± 0
ecc	14002 ± 582	24466 ± 510	14000 ± 0
ex4p	58678 ± 1513	75667 ± 14270	14000 ± 0
C2670	127713 ± 2805	172182 ± 35844	14000 ± 0
rot	153654 ± 2791	188909 ± 38129	14000 ± 0
x3	146850 ± 2571	247564 ± 4556	14000 ± 0
i7	170669 ± 4067	276803 ± 25262	14000 ± 0
frg2	192546 ± 2660	318533 ± 20008	14000 ± 0

[10] M. R. del Solar, J. A. G. Pulido, J. M. S. Perez, and M. A. V. Rodriguez, "Genetic algorithms for solving the placement and routing problem of an FPGA with area constraints," *Proc. IEEE ISDA*, pp. 31–35, 2004.

[11] S. N. R. Borra, A. Muthukaruppan, S. Suresh, and V. Kamakoti, "A novel approach to the placement and routing problems for field programmable gate arrays," *Applied Soft Computing*, vol. 7, no. 1, pp. 455–470, 2007.

[12] P. Jamieson, "Revisiting genetic algorithms for the FPGA placement problem." in *GEM*. Citeseer, 2010, pp. 16–22.

[13] Y. Xu and M. A. Khalid, "QPF: efficient quadratic placement for FPGAs," in *Field Programmable Logic and Applications, 2005. International Conference on*. IEEE, 2005, pp. 555–558.

[14] P. Gopalakrishnan, X. Li, and L. Pileggi, "Architecture-aware FPGA placement using metric embedding," in *Design Automation Conference, 2006 43rd ACM/IEEE*. IEEE, 2006, pp. 460–465.

[15] M. Xu, G. Gréwal, and S. Areibi, "Starplace: A new analytic method for FPGA placement," *INTEGRATION, the VLSI journal*, vol. 44, no. 3, pp. 192–204, 2011.

[16] T.-H. Lin, P. Banerjee, and Y.-W. Chang, "An efficient and effective analytical placer for FPGAs," in *Proceedings of the 50th Annual Design Automation Conference*. ACM, 2013, p. 10.

[17] A. Ludwin and V. Betz, "Efficient and deterministic parallel placement for FPGAs," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 16, no. 3, p. 22, 2011.

[18] B. M. Riess, K. Doll, and F. M. Johannes, "Partitioning very large circuits using analytical placement techniques," in *Design Automation, 1994. 31st Conference on*. IEEE, 1994, pp. 646–651.

[19] M. Gort and J. H. Anderson, "Analytical placement for heterogeneous FPGAs," in *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*. IEEE, 2012, pp. 143–150.

[20] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed *et al.*, "VTR 7.0: Next generation architecture and CAD system for FPGAs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 7, no. 2, p. 6, 2014.

[21] M. LGSynth93, "Benchmarks," *Obtained from http://www. eecg. toronto. edu/˜ lemieux/sega/ccts_blif. tar*.

[22] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for deep-submicron FPGAs*. Springer Science & Business Media, 2012, vol. 497.

[23] E. Cantú-Paz, "A survey of parallel genetic algorithms," in *Calculateurs paralleles*. Citeseer, 1998.

[24] E. Alba and M. Tomassini, "Parallelism and evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 5, pp. 443–462, 2002.

[25] P. Jamieson, "Exploring inevitable convergence for a genetic algorithm persistent FPGA placer," 2011.

[26] D. Lewis, E. Ahmed, D. Cashman, T. Vanderhoek, C. Lane, A. Lee, and P. Pan, "Architectural enhancements in Stratix-III and Stratix-IV," in *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*. ACM, 2009, pp. 33–42.

[27] M. I. Masud and S. J. Wilton, "A new switch block for segmented FPGAs," in *Field programmable logic and applications*. Springer, 1999, pp. 274–281.

[28] [Online], "Berkeley logic interchange format (BLIF)." Oct Tools Distribution 2, 1992.

[29] ——, "VTR tool, University of Toronto," http://https://github.com/verilog-to-routing/vtr-verilog-to-routing/, accessed: 2018-04-09.

[30] M. E. H. Pedersen and A. J. Chipperfield, "Local unimodal sampling," *HL0801 Hvass Laboratories*, 2008.

[31] [Online], "rand_s , MSDN Microsoft Website," https://msdn.microsoft.com/en-us/library/sxtz2fa8.aspx, accessed: 2018-04-09.

[32] J. P. Romano *et al.*, "Bootstrap and randomization tests of some nonparametric hypotheses," *The Annals of Statistics*, vol. 17, no. 1, pp. 141–159, 1989.

[33] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika*, vol. 52, no. 3-4, pp. 591–611, 1965.