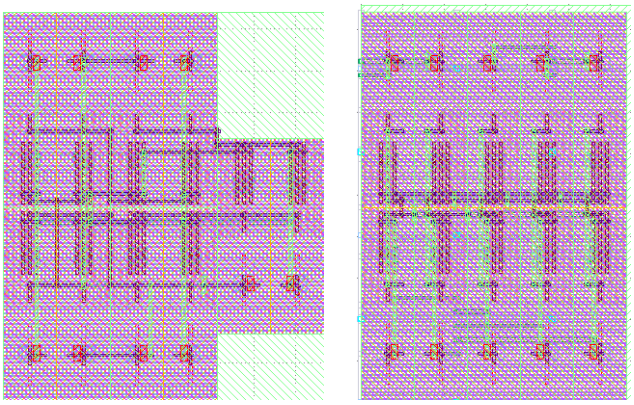
Self-Learning and Transfer across Topologies of Constraints for Analog / Mixed-Signal Circuit Layout Synthesis

Kaichang Chen, Georges G.E. Gielen
ESAT-MICAS, KU Leuven, 3001 Leuven, Belgium
Email: {kaichang.chen, georges.gielen}@kuleuven.be

***Abstract*—Truly full automation of analog/mixed-signal (AMS) integrated circuit design and layout has long been a target in electronic design automation. Making good use of human designer heuristics as constraints that steer today's tools is key to balancing efficiency and design space exploration. However, explicitly getting the constraints for every circuit from designers is the weak spot. Learning-based methods on the other hand can learn efficiently from training examples. This paper proposes a flexible framework that can self-learn various layout constraints for a circuit from some expert-generated example layouts. Constraints like alignment, symmetry, and device matching are learned from those expert layouts with the generate-and-aggregate methodology. Secondly, through feature matching, the learned knowledge can then be transferred as constraints for the layout synthesis of different circuit topologies, making the approach flexible and technology-agnostic. Experimental results show that our framework can learn constraints with 100% accuracy. Compared to other state-of-the-art tools, our framework also achieves a high efficiency and a high transfer accuracy over various types of constraints.**

***Index Terms*—constraint learning, analog and mixed-signal circuit layout synthesis, layout automation**



(a) layout generated by ALIGN without constraints (Area=18.1)

(b) layout generated by ALIGN with constraints (Area=15.4)

Fig. 1: Layout comparison of different layout generation approaches with relative indication of the area attained.

I. Introduction

Full automation of AMS integrated circuit sizing and layout has been a target in EDA for a long time [1]–[5]. AMS synthesis tools are confronted with huge solution spaces, with many degrees of freedom, facing a conflict between efficiency and optimality. To make the layout synthesis tractable and to have the tools generate layouts that meet all requirements, geometric constraints like symmetry and matching are typically imposed to steer the layout task, in the same way as layout experts use to do. For example, the manually generated layout of the ring oscillator is often aligned horizontally to occupy the minimum area. A state-of-the-art layout tool such as ALIGN [6] does not generate such a compact layout if not given the appropriate constraints (see Fig.1a). But after entering these constraints into the tool, a comparable layout is generated (shown in Fig.1b).

In state-of-the-art analog layout automation tools [6], [7], the user needs to explicitly enter the constraints for the design goal to be respected, such as symmetry groups, in a separate file. This requires a large expertise and effort from the user to do this correctly and completely. The lack of automated constraint generation has hindered the full adoption of AMS synthesis tools in the industry and therefore is a problem that needs to be solved [8]. Since the circuit netlist or schematic does not have all the required information to learn these constraints, a better alternative is to learn them directly from a few expert-created layouts of the circuit in combination with the schematic [9]. That is the focus of this paper.

This paper presents a new constraint learning methodology that can self-learn a variety of geometric constraints from some expert-generated layouts of the same topology. Using feature matching, the constraints are then transferred to different circuit topologies. The main contributions of this work are as follows:

- A constraint learning framework for AMS circuit layout automation is proposed, that can self-learn various types of geometric constraints from expert-generated example layouts. The proposed framework has good data efficiency, good runtime efficiency, and high accuracy.
- A two-step constraint transfer flow is introduced. It comprises feature matching on node embeddings and a searching-based completion step. The advantages include being agnostic to technology changes and handling hierarchical designs.
- Several searching-based algorithms are developed to complete the set of constraints. Experimental results show that the algorithms are very efficient at runtime; the completed transferred constraint group also achieves good accuracy among all types of constraints.

The remainder of the paper is organized as follows. In

979-8-3503-4859-0/DATE24/© 2024 EDAA

TABLE I: Comparison of the methodology with the state of the art.

	This work	ISVLSI'06 [9]	ISPD'10 [10]	DAC'21 [11]
constraint types	power net, alignment symmetry, configuration	symmetry, clustering, configurations	symmetry, matching, proximity	symmetry
circuit representation	bi-partite graph	sequence-pair	heterogeneous multigraph	heterogeneous multigraph with GNN embeddings
algorithm complexity	medium	medium	high	high
learning method	generate-and-aggregate	layout representation	netlist analysis with sub-block annotation	graph neural netowrks (GNN)
data requirement	one or more similar designs with expert layout	one layout of the same design in another technology	none	tens of different labeled designs

Section II, previous methodologies to learn or extract layout constraints will briefly be summarized. Section III will explain the proposed constraint learning and transfer framework. Section IV will present two experiments on transferring the learned knowledge. Section V will give the performance summary of the methodology and compare it to state-of-the-art methods, while Section VI will provide the conclusions.

II. Related Works

Several methods have been presented to extract a set of design constraints directly from the netlist analysis. For placement constraints, Michael Eick et al. [10] extracted constraints from the netlist topology with building block annotations and symmetry condition check. For routing constraints, sensitivities to parasitics obtained from netlist simulations have been used to form matching and bounding constraints [12]. The advantage of these methods is that no explicit input from the designer is required, but due to the complexity of the analog constraint extraction (ACE) problem, it is hard to say how these methods generalize to a large scale [13].

Learning-based approaches have also been investigated to learn AMS circuit layout constraints, especially symmetry constraints [11], [14]–[16]. In this type of method, a similarity score is computed from each pair of devices on the netlist to determine every possible inter-symmetrical pair. After training on a labeled dataset, an accuracy of over 90% has been achieved [11]. There is still concern about the risk of over-constraining the layout engine by those misclassified constraints, and it's not clear how to generalize the method to other types of constraints. Therefore, a more general method for learning the constraints with less possibility of over-constraining is needed, as will be presented in this paper. A comparison of different methodologies in the field to our work is summarized in Table I.

III. Constraint learning-and-transfer methodology

A. Preliminaries

Constraint learning, as a sub-field of constraint acquisition [17], has been introduced to build a combinational model directly from examples of positive or negative solutions for constraint programming problems. Similarly, our constraint learning-and-transfer framework for AMS circuit layout synthesis is used to reduce the design space for layout engines by extracting constraints from some layout examples generated by experts.

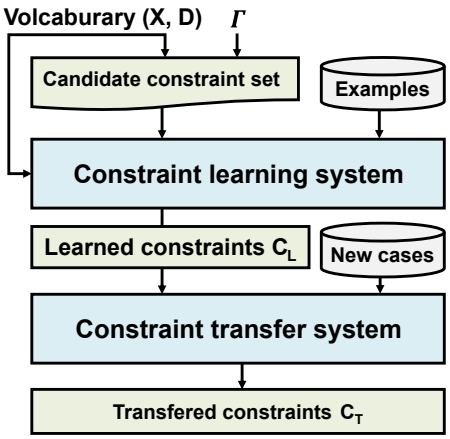


Fig. 2: Flow of the proposed constraint learning and constraint transfer framework.

The composition of our constraint learning and transfer framework is shown in Fig. 2. The constraint $Vocabulary(X, D)$ is formed by the expressions of a set of variables, X, in their domain D^X, and with an expressive constraint language Γ. With this information, possible candidate sets of constraints, $B = \{c_1, c_2, c_3, ..., c_{|B|}\}$, are tried on the problem. By giving the feasibility of these constraint sets queries, E, the target constraints, C_L, can be learned based on the system feedback. Then, the transfer system will try to generate a new constraint group, C_T, based on the features of C_L. The detailed operation of this framework will be explained below.

For the learning part, we use passive learning (from example solutions, i.e. from expert-created example layouts), in which the learning happens by giving the feasibility of constraints to each example solution. During this learning process, a feasible candidate set is chosen, and trivial constraints are also eliminated during the testing or aggregation process [18], [19].

B. Problem formulation

In the context of AMS circuit layout constraint learning, the variable domain for a circuit composed of N devices is formulated as :

$$X = \{Param_i \mid \forall i \in [1, N]\} \tag{1}$$

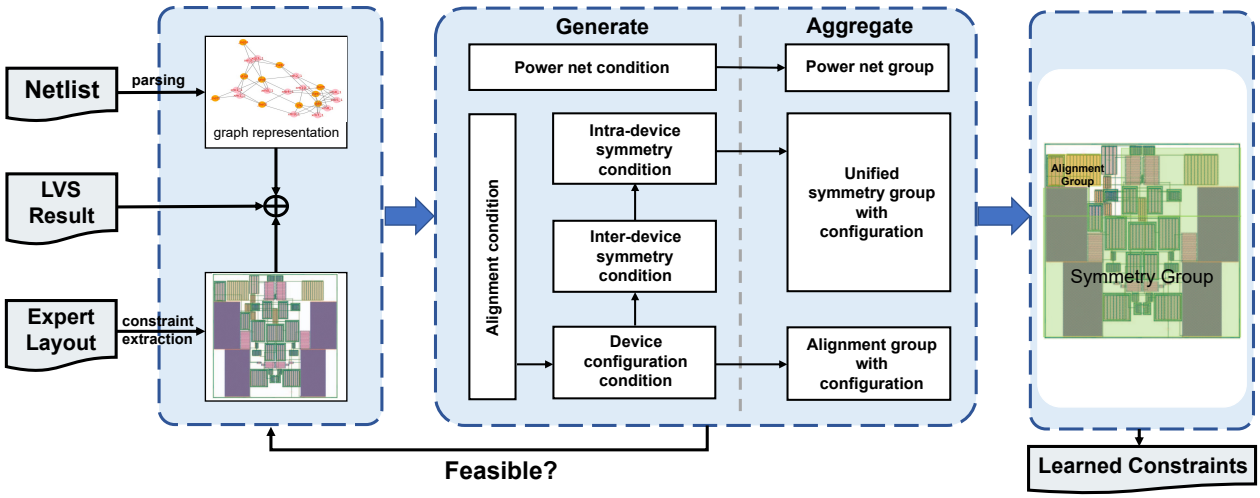


Fig. 3: General flow of the constraint learning system for AMS circuit layout synthesis.

In the variable domain X, the parameters of each device, $Param_i$, with their variable domains, are given by:

$$\begin{aligned} Param_i &= \{(x_i, y_i), refCell_i, angle_i, mirrorX_i, net_i\} \\ D^{x_i}, D^{y_i} &\in \mathbb{R}, D^{angle_i} \in [0, 360°], refCell_i \in D^{refCell} \\ D^{mirrorX_i} &\in \{0, 1\}, net_i \in nets \end{aligned} \quad (2)$$

In eq. (2), (x_i, y_i) denotes the central coordinates of components on the layout, and $refCell_i$ stands for the reference cell information of the component. $angle_i$ means the rotation angle of the component from the original reference cell. $mirrorX_i$ is a boolean variable to decide if the component is mirrored in the x-axis, and $D^{refCell}$ is limited to the reference cells that are used and stored in the layout.

The expressive language to connect these parameters in the variable domain into constraint sets is defined as:

$$\Gamma = \{>, <, =, \cup, +, -, \mid x \mid, \neg, \cap\} \quad (3)$$

In eq. (3), $>, <, =, +, -$ are of same meanings as in arithmetic. $\neg$ is the logical negation. $\cup$ and $\cap$ are intersection and union sets, respectively. $\mid x \mid$ is to get the absolute value of x. An example of the inter-device symmetry constraint candidate at the y-axis is formulated as, $B = \{y_i = y_j, mirrorX_i \neq mirrorX_j, refCell_i = refCell_j \mid i, j \in [1, N]\}$. By iterating through the possible candidate constraints, the true constraint group within the layout can be learned.

C. Learning from expert layouts

The learning of our framework is conducted with the flow shown in Fig. 3 for geometrical constraints like alignment, symmetry, device configurations, etc. To make the learning efficient, the query is generated from the most basic ones to more complicated ones. At the same time, infeasible decision variables are eliminated from the candidate set to help the system converge quickly.

In the flow of Fig. 3, the expert layout is first linked with its netlist representation through the information from the Layout Versus Schematic (LVS) tool. Then, this information is fed into the constraint learning system. From the defined expressive constraint language, Γ, and $Vocabulary(X, D)$, a series of possible constraint sets are generated.

TABLE II: Constraint conditions.

constraint type		conditions
power net		$\text{net}_i^{bulk} \cap Pins$
alignment		$\text{x}_i = x_j \cup \text{y}_i = y_j$
symmetry	inter device	$\text{mirror}_i = \neg mirror_j$
		$\text{refcell}_i = refcell_j$
	self	$\text{x}_i = x_{sym} \| \| y_i = y_{sym}$
device configuration	diffusion sharing	$(\text{L}_i + L_j) > \|2 * (x_i - x_j)\| \cup (\text{H}_i + H_j) > \|2 * (y_i - y_j)\|$

During the generate phase, the central coordinate of each variable, (x_i, y_i), is checked to identify alignment constraints. Then, the $angle_i$ and $mirrorX_i$ variables are checked within the domain of the alignment constraints to decide if an inter-device symmetrical constraint exists. If so, self-symmetrical conditions are checked alongside the symmetry axis. At the same time, after an alignment constraint is found, device configuration constraints are also checked, such as diffusion sharing, if the condition is feasible, the two constraints will be bonded together. The conditions for all these constraint types are listed in Table.II.

When all these kinds of constraints are generated, the constraints within the same group are aggregated together for simplicity and efficiency before finishing.

D. Using the learned knowledge

To apply the learned knowledge to new circuit topologies, a matching process must be conducted based on node embeddings. Table.III summarizes the features and dimensions of each type of node in its embedding. The device embedding is a 6-dimensional vector in total, while the net embedding is a 5-dimensional vector. During the matching phase, all dimensions of these vectors except for the constraint type are compared.

As shown in Fig. 4, the embedding of each node in the new design is firstly generated without any constraint during the

TABLE III: Node embedding.

Type	Feature	Description	Dims.
Device	active device flag	mark transistors and sub circuits	1
	node degree	# connected nets	1
	connection on ports	# connections for each port	3
	constraint type	mark constraint of the node	1
Net	net type	mark pin and power nets	1
	node degree	# connected devices	1
	connection on ports	net type on each port	3

matching process. Then, the node embeddings in the new design are compared with those in the learned group. If any node embedding of net type matches, device neighbors of matched nodes are also checked to derive the new matched constraint group. After a positive device matching, the constraint on the learning device embedding is added to the embedding of the matched device. After this process, the matched constraints will be used as starting points to learn the complete constraint set.

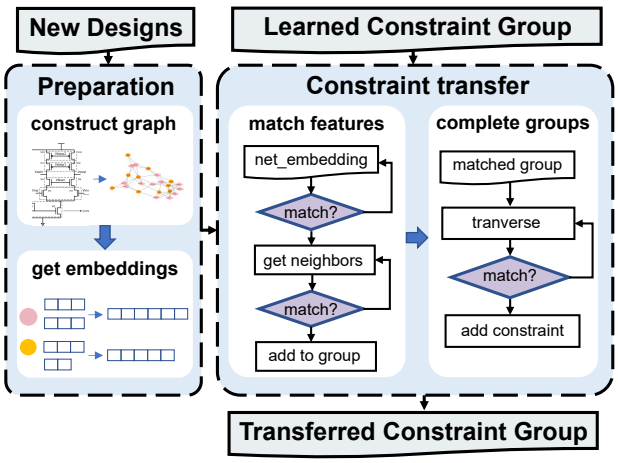


Fig. 4: General flow of the constraint transfer system.

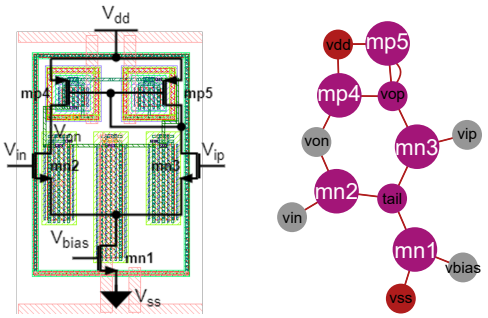
IV. Experimental Results

This section presents examples of some common constraint learning cases and a detailed process of how our proposed constraint learning and transfer framework addresses them.

A. Partial matching

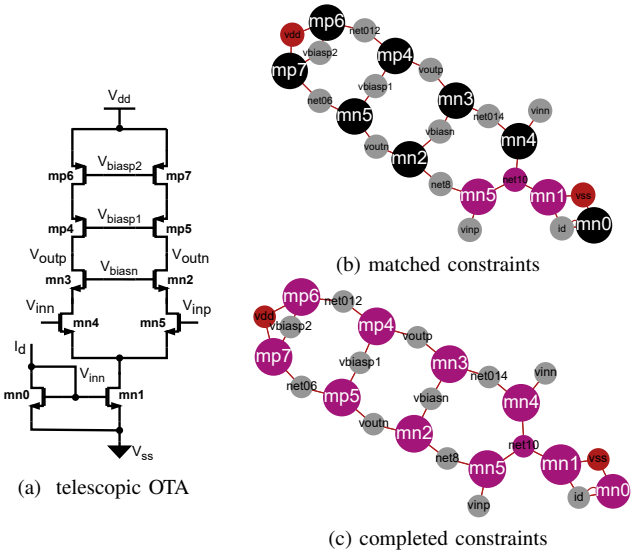
Partial matching can happen due to the complexity and diversity of the new topologies. In such a case, the matched embedding is very few. To have a stable embedding, only exactly the same devices are added with reference constraints. Fig. 5 shows the learning of a symmetry group from an operational amplifier (OTA) design (shown in Fig.5a), which forms a unified symmetrical device group, marked in pink in the graph representation. The power nets are also learned and marked as red among the net nodes (shown in Fig. 5b).

The learned symmetry constraint group is then used for transfer to a telescopic OTA (shown in Fig. 6a). As shown in Fig. 6b, only two devices, mn1 and mn5, are matched initially because only these nodes have the same connection type as in the learned constraint group. This incompleteness then leads to the following completing step.



(a) reference OTA (b) constraints

Fig. 5: Constraint learning on a five-transistor OTA.



(a) telescopic OTA (b) matched constraints (c) completed constraints

Fig. 6: Constraint transfer to a telescopic OTA.

Algorithm 1 BFS-based symmetry group completion.

Require:
Circuit Graph $G(V, E)$, power set P
Starting points $V\{Sym_{inter}, Sym_{self}\}$
Ensure:
Completed Group V'$\{Sym_{inter}$,$Sym_{self}\}$
let Q_0, Q_1 be queues
starting points to Q_0 or Q_1 based on sides
while $Q_0 \neq empty$ and $Q_1 \neq empty$ **do**
  BFS search on unvisited device neighbors of Q_0, Q_1
  for all unvisited neighbor product $item_{pair}$ **do**
    if $item_{pair}$ are the same type and connection **then**
      add $item_{pair}$ to Sym_{inter} and mark as visited
    else if $item_{pair}$ are identical and connection **then**
      add $item_{pair}$ to Sym_{self} and mark as visited
    end if
  end for
end while

In this completion step, a search process is conducted from the matched constraint group. Algorithm 1 shows the process

TABLE IV: Constraint learning results.

circuit type	technology	# net	# device	# hierarchies	learning accuracy (correctly identified / total)				runtime (sec)
					power net	configuration*	alignment group	symmetry group	
rail-to-rail comparator	130nm	17	26	1	1.00	26/26	18/18	4/4	2.18
voltage controlled oscillator (VCO)	40nm	7	10	2	1.00	-†	2/2	-†	2.01

*: The configuration constraints here means diffusion sharing for device matching

†: This type of constraint is not found on the expert design we used

TABLE V: Constraint transfer results.

circuit type	technology	# net	# device	# hierarchies	transfer F_1 score after completion				runtime (sec)
					power net	configuration	alignment group	symmetry group	
high-speed comparator	14nm	17	26	1	1.00	1.00	1.00	1.00	0.97
strongArm comparator	40nm	11	10	1	1.00	1.00	1.00	1.00	0.87
clocked comparator	60nm	14	12	2	1.00	1.00	0.80	1.00	1.07
cascode dynamic comparator	130nm	24	18	1	1.00	1.00	1.00	1.00	1.16
industrial voltage regulator	350nm	23	30	1	1.00	1.00	1.00*	1.00*	1.02
delta-sigma ADC	80nm	161	191	12	1.00	-†	-†	0.72	1.40
current starved VCO	14nm	11	14	2	1.00	-	1.00	-	1.08

* : Single inter-symmetry pairs here are also classified as alignment constraints

†: No known ground truth for this metric

used to get the complete group (shown in Fig. 6c). This algorithm is based on breadth-first search (BFS). The key change is that the search stops at power nets.

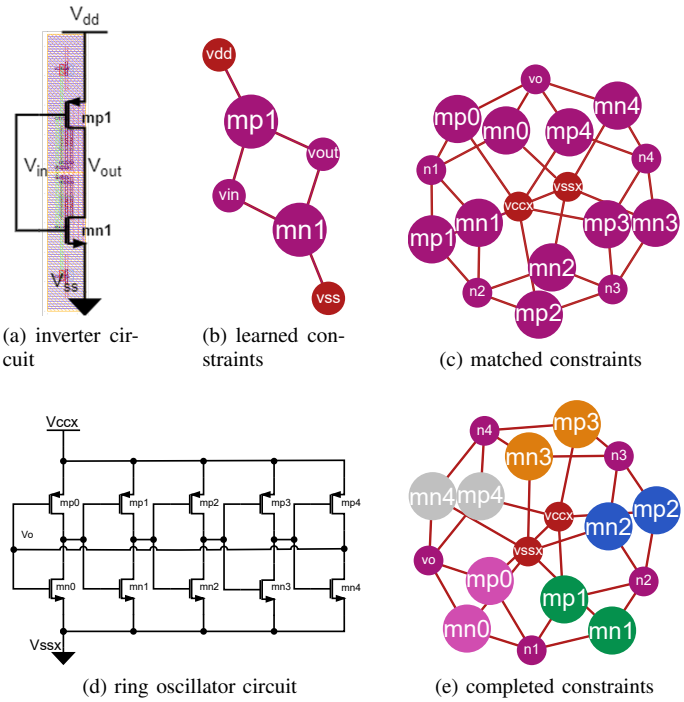
B. Multi-usage of constraints

It is also important to deal with multiple instances of the same constraint, such as multiple symmetry axes [16]. To demonstrate how our framework works in this case, an example of the multi-usage of an alignment constraint learned from an inverter is given here (shown in Fig. 7).

From an inverter design (Fig. 7a), an alignment constraint is extracted. The constraint is marked as purple in its graph representation, which has an alignment feature composed of two common nets (shown in Fig. 7b). Then, approximate embedding matching is used to match the corresponding instances of the group on a ring oscillator (shown in Fig. 7d). Approximate matching means that if an embedding, V_{new}, is an approximate matching of the embedding, $V_{original}$, every bit of V_{new} is equal to or greater than that of $V_{original}$, as given in eq. (4).

$$V_{new}[i] \geq V_{original}[i], \forall i \in \mathbb{Z} \wedge i \in [0, length] \quad (4)$$

After that, all possible constraint sets around these starting points are generated (shown in Fig.7c), which are then completed with Algorithm.2. During the completion step, all unvisited neighbors are checked and added to the visited group if the connection type matches (line 3). In this way, all valid instances of the reference alignment constraint are correctly extracted from the ring oscillator circuit (shown in Fig.7e).



(a) inverter circuit (b) learned constraints (c) matched constraints

(d) ring oscillator circuit (e) completed constraints

Fig. 7: Multi-usage of learned constraints on a ring oscillator.

Algorithm 2 Multi-usage group completion.

Require:
Circuit graph $G(V, E)$, Matched points $V\{align_l, align_r\}$
Template constraint group $Group_{ref}$
Ensure:
Completed groups V'
1: **for** $item_{pair}$ in between $align_l$ and $align_r$ **do**
2: **if** $item_{pair}$ matches $Group_{ref}$ and unvisited **then**
3: add $item_{pair}$ to V' and mark as visited
4: **end if**
5: **end for**

V. Performance Analysis

In Table IV and Table V, the performances of our framework on learning and transferring geometric constraints are listed for each experiment and for each constraint type, as well as the run time for each experiment. All these experiments are conducted on a Windows laptop with an Intel i7 2.8 GHz CPU and 16 GB of memory.

In terms of learning accuracy, the four types of potential constraints (power net, configuration, alignment and symmetry) are all 100% accurately learned from the expert-generated samples. These samples include both planar designs (rail-to-

rail comparator) and hierarchical ones (VCO). Moreover, the run time to learn constraints on each sample is about 2 seconds, which is very efficient.

In terms of the transfer quality, two test sets are given: 1)four different comparator topologies, one industrial voltage regulator design, and one analog-to-digital converter (ADC) in various technology nodes (14nm, 40nm, 60nm, 130nm, and 350nm) for the constraints learned on the comparator; 2) one current-starved VCO for the constraints learned on the VCO. The metric F_1 score is adopted for the evaluation of the true positive (TP), the false positive (FP), and the false negative (FN) constraints identified in the test sample. The F_1 score is defined in eq. (5).

$$F_1 score = \frac{2TP}{2TP + FP + FN} \tag{5}$$

Except for one comparator that has a false positive transfer, and the ADC that doesn't have enough learned features, all other identified constraints show a remarkable F_1 score.

An overall performance comparison has also been conducted, as shown in Table VI. It can be seen that our framework has better transfer capability in terms of technology nodes and design complexity. It also shows advantages in solution quality, efficiency, and constraint diversity. This work achieves a high average F_1 score of 0.98 over four constraint types with the shortest average runtime of 1.08 seconds compared to other works.

TABLE VI: Performance comparison of the presented method to state-of-the-art approaches.

	This work	TCAD'14 [20]	DAC'21 [11]
methodology	constraint learning	knowledge based synthesis	GNN
# learning samples	2	3	20
# transfer samples	7	1	20
technology node	**any**	same	same
hierarchical designs	**yes**	no	yes
# constraint types	**4**	- *	1
symmetry avg. F_1	**0.95**	1.00	0.88
overall avg. F_1	**0.98**	1.00	0.88
training time (sec)	**2.10**	- *	- *
avg. runtime (sec)	**1.08**	1.55	2.75

*: This metric can't be found in the paper

VI. Conclusion

A framework has been presented for the self-learning and transfer of geometric constraints for analog/mixed-signal circuit layout synthesis. For the learning task, the generate-and-aggregate methodology is used to extract four types of constraints from some expert-created layouts. For the transfer task towards other circuit topologies, feature matching is conducted on those circuits; then search-based algorithms are used to obtain the whole constraint group for both inter-block and intra-block constraints at all hierarchical levels. The principle of the methodology has been described and illustrated with two examples: inter-symmetry constraints between OTA topologies and alignment constraints between inverters and ring oscillators. Results on additional topologies have been presented in the experimental results section. These results confirm that the proposed methodology has wide applicability in terms of both design complexity and technology nodes. In addition, the methodology achieves good efficiency and high accuracy compared to the state of the art.

Acknowledgments

This work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement n° 101019982 - AnalogCreate). The authors would also like to thank speakers from the ACP Summer School 2023 for their insightful talks.

References

[1] G. Gielen and R. Rutenbar, "Computer-aided design of analog and mixed-signal integrated circuits," *Proceedings of the IEEE*, vol. 88, no. 12, pp. 1825–1854, 2000.

[2] V. der Plas et al., "AMGIE-A synthesis environment for CMOS analog integrated circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (IEEE TCAD)*, vol. 20, no. 9, pp. 1037–1058, 2001.

[3] M. Liu et al., "OpenSAR: An open source automated end-to-end sar adc compiler," in *Proc. 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–9, 2021.

[4] J. Lu et al., "Automatic op-amp generation from specification to layout," *IEEE TCAD*, vol. 42, no. 12, pp. 4378–4390, 2023.

[5] G. Gielen, "Analog synthesis 3.0: AI/ML to synthesize and test analog ICs: hope or hype ?," in *Proc. 2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*, pp. 1–1, 2023.

[6] K. Kunal et al., "Invited: ALIGN – open-source analog layout automation from the ground up," in *Proc. 2019 56th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–4, 2019.

[7] H. Chen et al., "MAGICAL: An open- source fully automated analog IC layout system from netlist to GDSII," *IEEE Design & Test*, vol. 38, no. 2, pp. 19–26, 2021.

[8] J. Scheible and J. Lienig, "Automation of analog IC layout: Challenges and solutions," in *Proc. of the 2015 Symposium on International Symposium on Physical Design (ISPD)*, pp. 33–40, 2015.

[9] T. Nojima et al., "Adaptive porting of analog IPs with reusable conservative properties," in *Proc. IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures (ISVLSI'06)*, pp. 6 pp.–, 2006.

[10] M. Eick, M. Strasser, H. Graeb, and U. Schlichtmann, "Automatic generation of hierarchical placement rules for analog integrated circuits," in *Proc. ISPD*, pp. 47–54, 2010.

[11] H. Chen et al., "Universal symmetry constraint extraction for analog and mixed-signal circuits with graph neural networks," in *Proc. DAC*, pp. 1243–1248, 2021.

[12] E. Malavasi et al., "Automation of IC layout with analog constraints," *IEEE TCAD*, vol. 15, no. 8, pp. 923–942, 1996.

[13] K. Zhu et al., "Automating analog constraint extraction: From heuristics to learning," in *Proc. 2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 108–113, IEEE, 2022.

[14] M. Liu et al., "S^3DET: Detecting system symmetry constraints for analog circuits with graph similarity," in *Proc. ASP-DAC*, pp. 193–198, 2020.

[15] X. Gao et al., "Layout symmetry annotation for analog circuits with graph neural networks," in *Proc. ASP-DAC*, pp. 152–157, 2021.

[16] K. Kunal et al., "A general approach for identifying hierarchical symmetry constraints for analog circuit layout," in *Proc. ICCAD*, pp. 1–8, 2020.

[17] C. Bessiere et al., "Constraint acquisition," *Artificial Intelligence*, vol. 244, pp. 315–342, 2017.

[18] Luc De Raedt et al., "Learning constraints from examples," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[19] D. C. Tsouros, T. Guns, and K. Stergiou, "Learning constraint models from data," in *Proc. AAAI 2023 Bridge on Constraint Programming and Machine Learning (CPML)*, 2023.

[20] Po-Hsun Wu et al., "A novel analog physical synthesis methodology integrating existent design expertise," *IEEE TCAD*, vol. 34, no. 2, pp. 199–212, 2014.