Learning Circuit Placement Techniques through Reinforcement Learning with Adaptive Rewards

Luke Vassallo, Josef Bajada
Dept. of Artificial Intelligence
University of Malta
Msida, Malta
{luke.vassallo.13, josef.bajada}@um.edu.mt

***Abstract*—Placement is the initial step of Printed Circuit Board (PCB) physical design and demands considerable time and domain expertise. Placement quality impacts the performance of subsequent tasks, and the generation of an optimal placement is known to be, at the very least, NP-complete. While stochastic optimisation and analytic techniques have had some success, they often lack the intuitive understanding of human engineers. In this study, we propose a novel end-to-end Machine Learning (ML) approach to learn fundamental placement techniques and use experience to optimise PCB layouts efficiently. To achieve this, we formulate the PCB placement problem as a Markov Decision Process (MDP) and use Reinforcement Learning (RL) to learn general placement techniques. The agent-driven data collection process generates highly diverse and consistent data points sufficient for learning general policies without expert knowledge under the guidance of an adaptive reward signal. Compared to state-of-the-art simulated annealing approaches on unseen circuits, the resulting policies trained with TD3 and SAC, on average, yield 17% and 21% reduction in post-routing wirelength. Qualitative analysis shows that the policies learn fundamental placement techniques and demonstrate an understanding of the underlying problem dynamics. Collectively, they demonstrate emergent collaborative or competitive behaviours and faster placement convergence, sometimes exceeding an order of magnitude.**

***Index Terms*—circuit layout, placement, reinforcement learning**

I. Introduction

PCBs are essential to modern electronics, providing a surface for soldering components and the routing infrastructure that connects them. PCB design translates a circuit's logical topology into a physical, manufacturable representation. This complex process involves precisely locating and orienting components on a specified board, and routing connections in line with the circuit schematic, all while adhering to manufacturing constraints. It's an iterative, demanding task requiring deep knowledge of components' operational constraints and physics.

PCB physical design has many commonalities with that of Integrated Circuits (ICs). Driven by the reliance on advanced automated Computer-Aided Design (CAD) tools for the fabrication of dense, high-performance ICs, the latter has garnered increased attention recently [1]. ML contributions are also becoming increasingly significant [1], demonstrating potential in assisting designers, reducing time to market, and enhancing design quality. However, PCB layout is predominantly a manual task, with automation research largely confined to addressing specific challenges [3], [5], [14].

Our goal is to use RL to learn effective placement techniques for solving the PCB component placement task end-to-end. Leveraging experience to optimise circuit layouts while demonstrating an understanding of the fundamental problem dynamics are the most distinctive aspects of this work. Source code, datasets and pre-generated results are available on GitHub: https://www.github.com/lukevassallo/rl_pcb.git.

II. Background

The input to the placement task is a circuit representation, such as a *netlist*. The netlist contains comprehensive information about each component, including their sizes, pin locations, and how they connect to their neighbours. The netlist can be represented as a hypergraph $H = (V, E)$, where the nodes $V = \{v_1, ..., v_n\}$ are component *pins* and the hyperedges $E = \{e_1, ..., e_m\}$ correspond to *nets*. A net $e = \{v_i, ..., v_j\}$, where $|e| \geq 2$, connects two or more nodes. Each node belongs to a component, given by the function $c : V \rightarrow C$, where C is the set of all netlist components. As placement is proven to be NP-Complete [15], guaranteeing optimality becomes infeasible as circuit complexity increases. Hence, this task is typically formulated as a multi-objective optimisation problem in terms of overlap and wirelength, with the aim of minimising them.

In our approach, we utilise two formulations for wirelength to promote distinct placement behaviours. The Half-Perimeter Wirelength (HPWL) corresponds to the sum of all half-perimeter lengths of all quadrilaterals, each enclosing all nodes of each hyperedge. HPWL is defined in Equation 1, where (x_i, y_i) are the Cartesian coordinates of $v_i \in V$.

$$HPWL = \sum_{e \in E} \max_{\{v_i, v_j\} \subseteq e} |x_i - x_j| + \max_{\{v_i, v_j\} \subseteq e} |y_i - y_j| \quad (1)$$

Since each component can have more than one pin, $v(a, e) = \{v \in e | c(v) = a\}$ represents all the nodes in $e \in E$ that also belong to the same component $a \in C$. $cp(e) = \{\{c(v_i), c(v_j)\} | \{v_i, v_j\} \subseteq e \wedge c(v_i) \neq c(v_j)\}$ gives all the distinct unordered component pairs in net $e \in E$. The minimum distance between two components a and b for net e is defined in Equation 2, while Equation 3 defines the Euclidean Wirelength (EW), which sums all the minimum distances between distinct components of each net.

$$d(a,b,e) = \min_{v_i \in v(a,e), v_j \in v(b,e)} \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \quad (2)$$

$$EW = \sum_{e \in E} \sum_{\{a,b\} \in cp(e)} d(a,b,e) \quad (3)$$

III. Related Work

State-of-the-art placement techniques are driven by IC design due to their intractability to manual placement and are based on non-linear analytical techniques [8]. Unlike in IC design [2], [8], PCB physical design frequently employs meta-heuristic techniques like genetic algorithms [4], swarm intelligence [3], and simulated annealing [9], [12]. This preference arises because PCB circuit netlists are relatively smaller, rendering stochastic optimisation feasible and negating the need for complex differential cost function formulation. Placement co-optimisation receives more attention in literature, focusing on aspects like thermal management and power efficiency [3], electromagnetic compatibility [14], and the effects of parasitics on power converter efficiency [5].

Machine Learning (ML) is significantly changing the process of Electronic Design Automation (EDA) flow in three key areas [1]. Firstly by predicting performance metrics (e.g. wirelength, routability, power), thereby replacing time-consuming measurement processes with estimates. Secondly leveraging knowledge from past data for more efficient design space exploration [10], and finally AI-assisted workflows that automate entire steps [11]. Contemporary works tackling floorplanning or placement combine stochastic optimisation with ML [17] and RL [10]. An end-to-end approach was taken by [11] where the constructive floorplanning task was formulated as an MDP, and RL was subsequently used to place netlist elements onto a discretised IC canvas sequentially. However, their method lacks generalisation, particularly evident from the fine-tuning process prior to evaluation. Independent researchers also noted similar observations [10], and recent work has shown that the performance gains did not hold on open datasets [18].

IV. Method

We frame the iterative PCB component placement problem as an MDP [16]. In our approach, an agent represents a component on the PCB that can perceive the surrounding environment and take actions to adjust its position or orientation. The agent's objective is to minimise wirelength, while avoiding overlap as it reaches an equilibrium position. One component in the netlist is locked and serves as an anchor, while all other components are movable and subject to the policy in each episode step. Therefore for a netlist of C components, this results in $C-1$ movable components and thus, the policy is invoked $C-1$ times during a single episode step.

A. Observations

An observation is a 23-element vector consisting of three feature sets; 16 elements capture the perceived surroundings, four elements capture direction information related to movement, and the remainder is the component's position and orientation.

We employ image processing techniques on a stack of grayscale images representing the circuit netlist to understand the immediate surroundings. Each component is assigned a distinct layer, drawn with a resolution of 0.5mm. Separately, we generate a set of masks around the current component by drawing a circle with a diameter 1.5 times its longest edge, centred and divided into eight segments. As illustrated by Figure 1a, a segment i serves as the basis for two masks: one for overlap detection, M_{Oi}, and the other for line-of-sight determination, M_{Li}, that is the detection of non-overlapping objects in the vicinity of the component. The overlap mask M_{Oi} for segment i is calculated in Equation 4a by applying pixel-wise binary AND operation between the circle segment, S_i and the current component, a. The line-of-sight mask M_{Li} comprises the pixels in segment S_i that do not correspond to the overlap mask M_{Oi} and is calculated by Equation 4b with a pixel-wise binary XOR between S_i and M_{Oi}.

$$M_{Oi} = S_i \ \& \ a \quad (4a)$$

$$M_{Li} = S_i \oplus M_{Oi} \quad (4b)$$

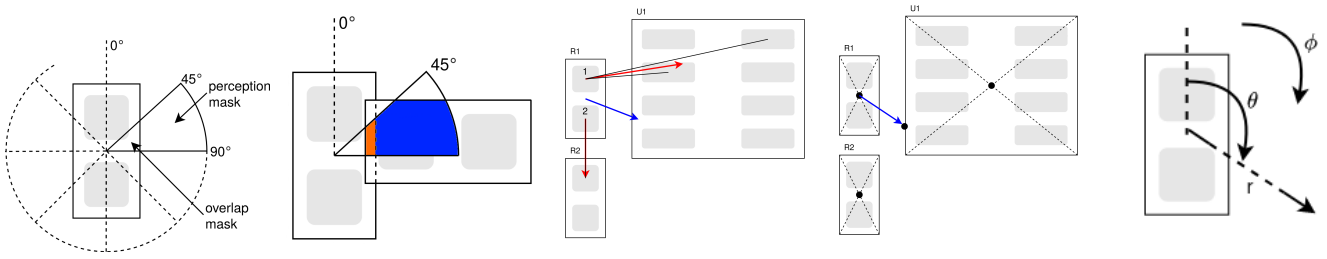
A bitwise AND operation is performed between the individual masks and all layers, excluding the current component, to derive the perception information. The mathematical operations for these calculations are respectively described in Equations 5a and 5b, where $C_j, \forall j \in N$ denotes the set of layers containing neighbour nodes of current node a. In Figure 1b, we present an example where a nearby component overlaps with the current component. The overlap contribution is in orange, while the non-overlapping contribution is in blue. These operations result in a normalised 16-element vector, with eight elements from overlap measurements and the remaining quantifying the presence of surrounding objects.

$$P_{Oi} = \frac{(M_{Oi} \ \& \ C_j, \forall j \in N)}{\sum M_{Oi}} \quad (5a)$$

$$P_{Li} = \frac{(M_{Li} \ \& \ C_j, \forall j \in N)}{\sum M_{Li}} \quad (5b)$$

Directional information is captured in two vectors of form (r, θ), termed goal-oriented and cluster-oriented. The goal-oriented vector points in the direction that minimises wirelength and is derived from all the pin-to-pin connectivity vectors. It is denoted in Cartesian form, (x_a, y_a), by Equation 6 where a denotes the current component, and N its neighbours and exemplified in Figure 1c. The black lines associated with pin 1 of component R1 are part of a multi-pin net. In this case, the resultant vector (drawn in red) is computed, and its magnitude is divided by the number of vectors involved in its calculation to maintain relatively small numerical values. Pin 2 of component R1 is a point-to-point connection; therefore, the vector is used as is. The blue vector results from adding the resultant vectors in red and yields the first feature.

The blue arrow in Figure 1d illustrates the cluster-oriented vector computed between the current component's centre and the group's centroid $(\bar{x}, \bar{y})$. The cluster's centroid is calculated over the current component and all its neighbours according to



(a) Perception Masks (b) Masking Example (c) Goal-oriented Vector (d) Cluster-oriented Vector (e) Action Space

Fig. 1: 1a depicts the mask generation from circle segments and 1b illustrates how overlapping (orange) and nearby non-overlapping (blue) features are captured. Figures 1c and 1d capture directional information relating to the overall direction of movement and identification of neighbouring components. 1e illustrates the continuous action space as a vector triplet.

the netlist. The feature vector (r_g, θ_g) is computed by Equation 7, between the current node's position (x_a, y_a) and the centroid. It provides information about the agent's relative position to its neighbours.

$$(x_a, y_a) = \sum_{e \in E} \sum_{v_i \in v(a,e)} \frac{1}{cp(e)} \sum_{v_j (N,e)} (x_i - x_j), (y_i - y_j) \quad (6)$$

$$r_g = \sqrt{(x_i - \bar{x})^2 + (y_i - \bar{y})^2}, \ \theta_g = \tan^{-1} \left(\frac{(y_i - \bar{y})^2}{(x_i - \bar{x})^2} \right) \quad (7)$$

Component information is the final piece of the observation that includes the Cartesian position (x_a, y_a), normalised by the board size and its orientation θ_a ranging between $-\pi^c$ to π^c.

B. Actions

A continuous action space is selected because of its ability to describe movement in any direction and offers granular control over a discrete representation. Figure 1e contains three elements accounting for component translation and orientation. The former is defined as an (r, θ) vector with the magnitude, r, taking unit values, while the direction, θ, is specified in the range of 0^c to $2\pi^c$. The component's orientation, ϕ, is also a continuous value in the range $[0, 1]$, albeit is interpreted discretely according to Equation 8.

$$orientation = \begin{cases} 0^\circ \ when \ 0 \leq \phi < 0.25 \\ 90^\circ \ when \ 0.25 \leq \phi < 0.5 \\ 180^\circ \ when \ 0.5 \leq \phi < 0.75 \\ 270^\circ \ otherwise \end{cases} \quad (8)$$

C. Rewards

The reward signal conveys the designer's goals to the agent through evaluative feedback after each episode step and consists of two elements. The first term is an early termination penalty that comes into effect only if the agent deviates from the problem scope. The quality of the action, directly linked to the overall solution quality, is captured by the second part. The agent can position the component anywhere on the Cartesian plane, representing the layout area, without any restrictions imposed by the environment. However, moving outside the board region triggers a premature episode termination with a penalty proportional to the remaining steps, as we aim to keep the agent's operations within the board. Thus, the episode lengths vary based on the agent's ability to operate within the board region, and as discussed in section IV-F, the agent learns this behaviour and begins completing full episodes after 75k steps. The early termination penalty, ζ, is formulated in Equation 9, where $T = 200$ is the total episode steps, $t \leq T$ denotes the agent's last step, and s_t is the state at t. $p_r = 8$ is a constant that scales the penalty for stopping before T.

$$\zeta = \begin{cases} p_r(T - t) & \text{if } s_t \text{ is terminal} \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

Credit is immediately assigned after each episode step according to Equation 10, which linearly combines the EW W_t (prioritising the agent over its cluster), HPWL H_t (rewarding clusters sharing common nets), and overlap O_t (ensuring a legal, overlap-free layout). O_t is detailed in Equation 11c, with P_{Oi} representing the normalised overlap contribution of segment i, as defined in Equation 5a. The tangent function aggressively rewards actions leading to overlap-free wirelength improvements, constrained within $[0, \frac{\pi}{2.1}^c]$. Differing from [11], overlap is not a hard constraint, allowing initial placement overlaps but incentivising their avoidance as the solution emerges in later stages. Weights n, m and m on these metrics control the policy's learned behaviour, balancing between independent component optimisation (EW emphasis) and collaborative net length minimisation across clusters (HPWL emphasis).

$$r_t = \tan \left(\frac{nW_t + mH_t + p(1 - O_t)}{n + m + p} \cdot \frac{\pi}{2.1} \right) - \zeta \quad (10)$$

The success of learning general behaviour partly arises from rewarding the agents indiscriminately across layouts. The wirelength contribution differs between clusters of components, and therefore to consistently reward the agent's actions, the wirelength needs to be normalised per net basis. With reference to Equations 11a and 11b for W_t and H_t respectively, this is achieved by normalising the value against the initial conditions, w_0 and h_0, and the best known historical values, w_e and h_e. The former establishes a baseline against which the agent's actions

can be rewarded. The latter represents the best historically known HPWL and EW for each net in a circuit netlist. These values may be extracted from human-generated layouts, or ballpark figures can be identified by performing a few episodes before training. In both scenarios, as the policy learns, better values for these parameters are identified until reaching near-optimal ones, which is when the reward signal will more accurately gauge the agent's performance. In other words, the policy will iteratively self-improve to an adapting reward signal.

$$W_t = clip\left(\frac{w_0 - w_t}{w_0 - w_e}, -1, 1\right) \tag{11a}$$

$$H_t = clip\left(\frac{h_0 - h_t}{h_0 - h_e}, -1, 1\right) \tag{11b}$$

$$O_t = \frac{1}{8}\sum_{i=0}^{7} P_{Oi} \tag{11c}$$

D. Dataset

We construct a dataset of nine unique seed circuits, each containing three to twelve components, and constrained to a board area of $400mm^2$. The circuits are derived from real-world designs and contain analog, mixed-signal and digital topologies. The training dataset M_T consists of six circuits, and the remainder comprises the unseen testing dataset, M_U. To the best of our knowledge, no publicly available PCB circuit dataset exists for physical design. All circuits are provided in KiCAD PCB format.

E. Experiments and Setup

We conduct several experiments to understand the effects of emphasising different aspects of the reward signal by adjusting the weights of EW, HPWL and overlap. In addition, we perform ablation tests by removing one of the wirelength terms (i.e. $n = 0$ or $m = 0$). Table I lists the configurations of the experiments, where each is executed four times using different seed values, deterministically derived from a user defined value arbitrarily set to 99. We report the average return and standard deviation computed for each experiment over all successful trials.

Policy optimisation is performed using advanced RL algorithms, notably Twin Delayed Deep Deterministic Policy Gradient (TD3) [6], which reduces overestimation bias through a dual Q-network, and Soft Actor-Critic (SAC) [7], known for balancing return and entropy for better exploration. To adapt to a changing reward signal, we employed a resizable replay buffer, initially set to a size of 25,000, that doubles after accumulating samples equal to twice its size. This strategy operates under the assumption that, in the initial phases, the reward signal is inconsistent because the policy quickly identifies better values for the wirelength parameters w_e and h_e. Therefore, the samples need to be recycled quickly. As the policy improves, the reward signal becomes stable, and the replay buffer size increases to accommodate more samples to learn from. The policy and Q network consist of a two-layer neural network with 400 and 300 neurons using a ReLU activation function. All remaining parameters default to those initially set by the authors for TD3 and SAC. Experiments were carried out on an Ubuntu 22.04 machine with a Core™ i7-10700K CPU, 64GB of RAM and a GeForce GTX 1080 GPU.

F. Evaluation

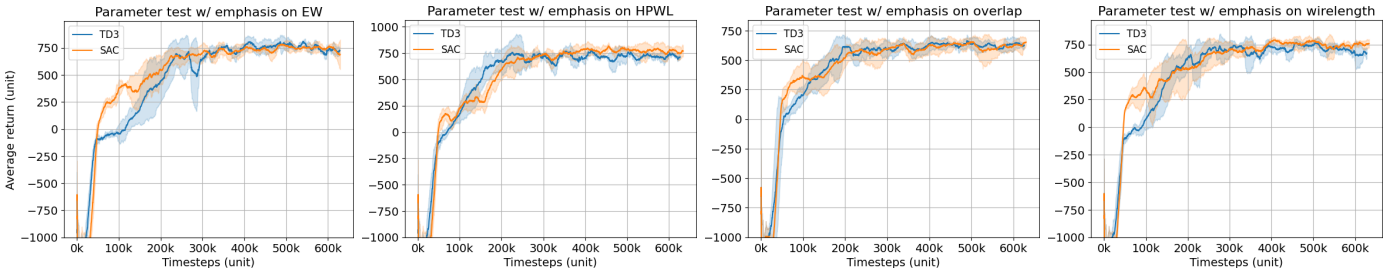
To evaluate, a random placement based on circuits in the unseen dataset M_U is generated and optimised separately using our policies and a simulated annealing placer, SA-PCB [12], over 500 steps. The optimised layouts are then routed using PcbRouter [13] with an A* routing algorithm, and the resulting post-routing wirelength is measured. This metric is preferred over estimates such as HPWL and EW since it considers the impact of placement quality on the routing process and potential issues like congestion. We run up to four trials on every circuit in M_U for every configuration and then calculate the average post-routing wirelength. SA-PCB runs for 500 iterations, while PcbRouter is assigned a high layer change cost of 100, forcing single-layer routing and ten rip-up and reroute iterations to identify an optimised route. Defaults are otherwise assumed.

V. Experimental Results

Table I summaries the average returns from all experiments, with configurations 1-4 further depicted in Figure 2. During a typical training process, the agent initially incurs substantial negative rewards, primarily from moving components outside the board region, leading to early episode termination. However, a notable improvement in the agent's performance is observed between 25k and 75k steps, as it quickly learns to stay within the board boundaries. This learning phase is marked by a sharp increase in rewards. Subsequently, the agent develops more complex behaviours for optimising the circuit layout, as indicated by a gradual, less steep rise in reward accumulation. This progression suggests that the agent not only masters the fundamental rules but also starts to grasp more nuanced strategies for maximising rewards that are aligned with our placement directives. Eventually, the reward signal stabilises, indicating a plateau in the learning curve.

The values in Table I reveal that SAC outperforms TD3 by over 10% in configurations that emphasise HPWL (reflected in the m coefficient). Conversely, TD3 shows a marginal advantage on configurations that prioritise EW (indicated by the n coefficient) and a greater advantage on configurations focused on minimising overlap (denoted by the p coefficient). SAC's better performance on a reward signal with a non-differential HPWL term and its lower variance during training can be attributed to its superior ability to maximise search space exploration and lower sensitivity to random initial conditions.

Table II summarises the routed wirelength resulting from optimising circuits in the M_U test dataset using the best policies, and the percentage improvement over SA-PCB baseline is recorded for both TD3 and SAC. In experimental setups that prioritise wirelength over overlap, the evaluation may not generate overlap-free results for all runs, and in such cases, the next best layout is selected, allowing for up to 10% overlap. This constraint is relaxed because future work can include a legalisation post-processing task to resolve minor overlaps. The average is computed over up to four evaluations.



(a) Emphasise EW (b) Emphasise HPWL (c) Emphasise Overlap (d) Emphasise Wirelength

Fig. 2: Average return for distinct policy training runs corresponding to experiment configurations 1-4 in Table I. All experiments are performed four times each initialised with deterministic random seed values. A moving average filter with a window of 100 samples (approximately 20,000 steps) smoothens the chart.

#	Configuration	TD3	SAC	% SAC
1	n=6,m=2,p=2	739.09 ± 354.98	750.20 ± 242.47	1.48%
2	n=2,m=6,p=2	715.00 ± 302.43	796.23 ± 270.45	10.2%
3	n=2,m=2,p=6	623.46 ± 272.96	612.65 ± 262.53	-1.77%
4	n=4,m=4,p=2	692.81 ± 348.48	743.65 ± 305.53	6.84%
5	n=0,m=5,p=5	510.22 ± 283.69	578.16 ± 232.07	11.75%
6	n=0,m=2,p=8	825.32 ± 257.57	780.23 ± 246.30	-5.78%
7	n=0,m=8,p=2	632.36 ± 366.32	720.11 ± 331.52	12.19%
8	n=5,m=0,p=5	729.98 ± 326.45	712.74± 367.55	-2.42%
9	n=2,m=0,p=8	1007.95 ± 294.59	942.54 ± 248.74	-6.94%
10	n=8,m=0,p=2	812.94 ± 366.38	802.84 ± 356.93	-1.26%

TABLE I: Average return and standard deviation over a maximum of four trials, for each experiment configuration.

#	Configuration	TD3			% TD3	SAC			% SAC
		M_{U0}	M_{U1}	M_{U2}		M_{U0}	M_{U1}	M_{U2}	
1	n=6,m=2,p=2	30.8	**13.3**	54.8	15.9%	30.9	13.9	55.3	14.3%
2	n=2,m=6,p=2	29.3	14.1	50.4	+17.4%	34.5	**10.6**	51.7	21.1%
3	n=2,m=2,p=6	36.7	16.2	63.8	-0.1%	40.9	15.6	62.3	-2.0%
4	n=4,m=4,p=2	**27.9**	14.2	56.1	15.7%	36.4	12.9	**51.4**	13.9%
5	n=0,m=5,p=5	45.0	15.9	61.8	-5.6%	38.1	15.6	59.6	2.0%
7	n=0,m=2,p=8	44.9	15.6	86.0	-15.4%	59.6	16.3	73.8	-24.5%
6	n=0,m=8,p=2	29.0	15.5	46.0	16.0%	**28.7**	15.2	50.8	15.0%
8	n=5,m=0,p=5	35.5	16.9	64.4	-1.3%	36.1	15.5	61.3	3.3%
9	n=8,m=0,p=2	32.7	15.5	**42.7**	14.3%	29.3	15.1	58.4	11.3%
10	n=2,m=0,p=8	43.2	15.4	71.5	-7.1%	48.3	16.4	71.7	-14.0%
	SA-PCB	38.9	**13.3**	75.3		38.9	13.3	75.3	

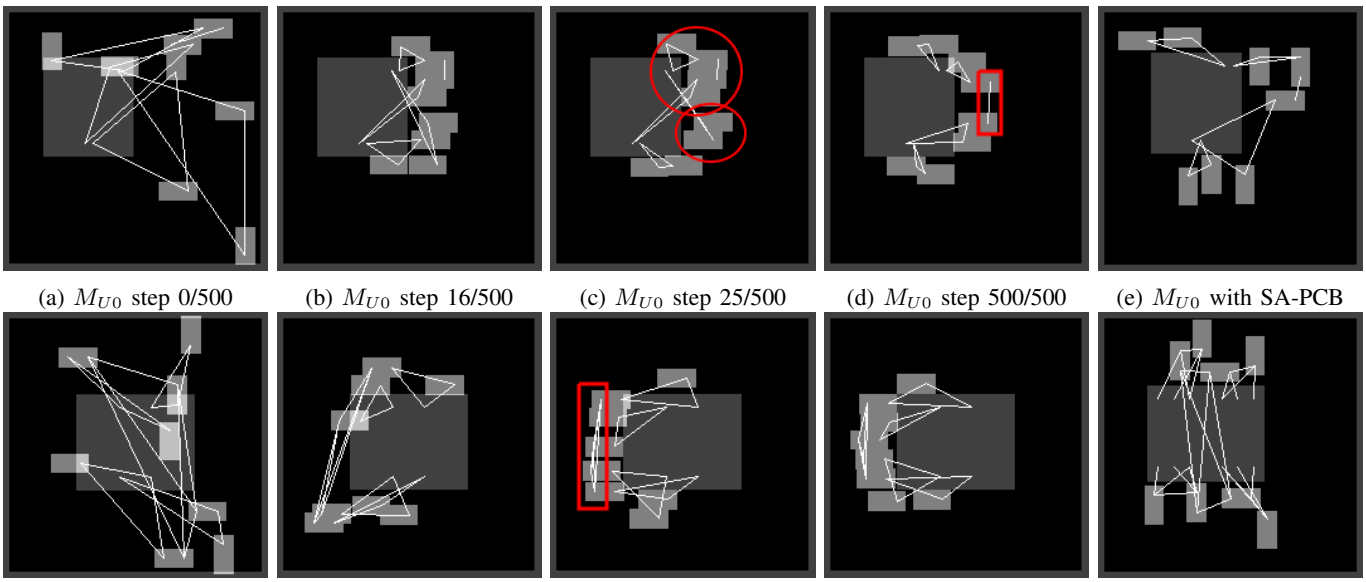
TABLE II: Post-routing wirelength generated by the best policy for each configuration averaged over a maximum of four trials, compared with layouts generated with SA-PCB using default settings. All wirelength values are generated by PcbRouter [13].

The results of the first four configurations suggest that policies prioritising overlap tend to produce layouts with higher wirelength but have a greater likelihood of generating overlap-free results while yielding roughly the same wirelength as SA-PCB. Placing more emphasis on HPWL generates layouts with lower wirelength than SA-PCB. In particular, the second configuration $(n = 2, m = 6, p = 2)$ trained with SAC outperformed SA-PCB on all unseen layouts, resulting in an average wirelength reduction of 21.1%, while TD3 on the same configuration achieved a slightly lower improvement of 17.4%. Configurations 1-4 tend to meet and exceed the performance of SA-PCB, regardless of whether they were trained with TD3 or SAC. Policies generated from ablation experiments ($m = 0$ or $n = 0$) that assign increased weight to overlap are significantly outperformed by SA-PCB. In contrast, policies trained by both TD3 and SAC that prioritise HPWL $(n = 0, m = 8, p = 2)$, on average, outperformed SA-PCB by 16% and 15%, respectively, and those that prioritise EW $(n = 8, m = 0, p = 2)$ by 14.3% and 11.3%, respectively. Promoting collaborative behaviours by rewarding HPWL generates better post-routing wirelength, as components tend to have nets spanning more than two pins.

VI. Policy Analysis

Qualitative analysis is presented in this section by tracing the key steps from random initialisation until the optimised layout emerges. For circuit M_{U0}, the initial placement for a policy greedily optimising HPWL $(n = 0, m = 8, p = 2)$ is shown in 3a. While the components quickly move towards the anchoring IC in 3b, the result is a poor placement that requires drastic changes before yielding an optimised layout. Several swaps occur reaching 3c that require the agents involved to temporarily undergo a sequence of actions that yield low rewards until a better placement is reached. Examples of such moves require circling around neighbouring components, temporarily increasing wirelength, or exchanging positions while tolerating some overlap. This process is repeated a couple of times for components in the marked clusters to exchange their position over the next 325 steps, which is when the final placement emerges in Figure 3d. The placement yielded by SA-PCB is depicted in Figure 3e, and while component clusters emerge, the placement is not optimised, evident from the poor placement of clusters residing at the top-left and centre-right of the layout.

In a different scenario, an initial placement for a neutral policy $(n = 0, m = 5, p = 5)$ is shown in Figure 3f. The components quickly divide into two groups depending on their connections to the main IC, as Figure 3g shows. Subsequent moves evolve into a single large cluster linked by the dominant multi-pin net and several smaller ones. After just 29 steps, the final placement resembles its final positioning, as depicted in the terminal state by Figure 3i. Circuit M_{U2} provides a challenging scenario because most components simultaneously form part of a large cluster and multiple smaller ones. SA-PCB, in general, performs poorly on this circuit, as demonstrated by Table II. Moreover, after 500 invocations, the layout shows that the placement is divided between optimising for the large



(a) M_{U0} step 0/500 (b) M_{U0} step 16/500 (c) M_{U0} step 25/500 (d) M_{U0} step 500/500 (e) M_{U0} with SA-PCB

(f) M_{U2} step 0/500 (g) M_{U2} step 14/500 (h) M_{U2} step 29/500 (i) M_{U2} step 500/500 (j) M_{U2} with SA-PCB

Fig. 3: Policy behaviour analysis. 3a-3d illustrate key moments when optimising layout M_{U0} using a policy emphasising HPWL $(n = 0, m = 8, p = 2)$. Alongside it in 3e is the result from SA-PCB starting from the same initial conditions. Similarly, 3f-3i show layout M_{U2} being optimised with a neutral policy $(n = 0, m = 5, p = 5)$ alongside the result by SA-PCB in 3j.

cluster and the individual smaller ones, resulting in a placement of relatively low quality. These examples suggest that guiding actions based on experience yields both quicker and better results over blind attempts with the hope of improving quality.

VII. Conclusion

As ML is integrated into EDA workflows, methods that robustly leverage experience for solving physical design tasks become increasingly valuable for AI-assisted flows or no-human-in-loop design. Our novel methodology formulated the iterative component placement task as an MDP and used RL for optimising layouts end-to-end. We collected diverse training data by allowing all components in the netlist to contribute different perspectives and guided by an adaptive reward signal learned general policies. Our results demonstrated competitive performance compared to simulated annealing on post-routing wirelength, and the learned policies showed a good understanding of the task dynamics. Although further optimisations and key features are necessary, this work opens up promising avenues for future research towards automating PCB placement.

References

[1] B. Khailany et al., "Accelerating Chip Design With Machine Learning," IEEE Micro, vol. 40, no. 6, pp. 23–32, Nov. 2020.

[2] I. L. Markov, J. Hu, and M.-C. Kim, "Progress and Challenges in VLSI Placement Research," Proc. IEEE, vol. 103, no. 11, pp. 1985–2003, Nov. 2015.

[3] A. Alexandridis, E. Paizis, E. Chondrodima, and M. Stogiannos, "A particle swarm optimization approach in printed circuit board thermal design," ICA, vol. 24, no. 2, pp. 143–155, Mar. 2017.

[4] T. Badriyah, F. Setyorini, and N. Yuliawan, "The implementation of Genetic Algorithm and Routing Lee for PCB design optimization," in 2016 International Conference on Informatics and Computing (ICIC), Mataram, Indonesia: IEEE, 2016, pp. 148–153.

[5] P. Ning, H. Li, Y. Huang, and Y. Kang, "Review of power module automatic layout optimization methods in electric vehicle applications," Chin. J. Electr. Eng., vol. 6, no. 3, pp. 8–24, Sep. 2020.

[6] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in International conference on machine learning, 2018, pp. 1587–1596.

[7] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in International conference on machine learning, 2018, pp. 1861–1870.

[8] Y. Lin, S. Dhar, W. Li, H. Ren, B. Khailany, and D. Z. Pan, "DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement," in Proceedings of the 56th Annual Design Automation Conference 2019, Las Vegas NV USA: ACM, Jun. 2019, pp. 1–6.

[9] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," Science, vol. 220, no. 4598, pp. 671–680, 1983.

[10] Q. Xu et al., "GoodFloorplan: Graph Convolutional Network and Reinforcement Learning-Based Floorplanning," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 41, no. 10, pp. 3492–3502, Oct. 2022.

[11] A. Mirhoseini et al., "A graph placement methodology for fast chip design, Nature, vol. 594, no. 7862, pp. 207–212, 2021.

[12] C. Holtz, D. J. Merrill, and M. Woo, SA-PCB: Simulated Annealing-based Placement For PCB Layout. GitHub, 2020. [Online]. Available: https://github.com/The-OpenROAD-Project/SA-PCB

[13] T.-C. Lin, C. Holtz, Yenyi, and D. J. Merrill, The OpenROAD Project - Printed Circuit Board (PCB) router. GitHub, 2020. [Online]. Available: https://github.com/The-OpenROAD-Project/PcbRouter

[14] P. Makeev, "Two-level algorithm for automated placement of elements on a flex-rigid printed circuit board," in 2021 International Conference on Electrotechnical Complexes and Systems (ICOECS), 2021, pp. 196–201.

[15] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified NP-complete graph problems," Theoretical Computer Science, vol. 1, no. 3, pp. 237–267, 1976.

[16] C. Boutilier, "Planning, learning and coordination in multiagent decision processes," in TARK, 1996, vol. 96, pp. 195–210.

[17] Y.-H. Huang et al., "Routability-Driven Macro Placement with Embedded CNN-Based Prediction Model," in 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2019, pp. 180–185.

[18] C.-K. Cheng, A. B. Kahng, S. Kundu, Y. Wang, and Z. Wang, Assessment of Reinforcement Learning for Macro Placement. 2023.