Proceedings Paper:
Sun, J. orcid.org/0009-0007-0540-7482, Zha, X. orcid.org/0000-0002-4616-2334, Wang, C. orcid.org/0009-0000-1248-889X et al. (4 more authors) (2024) Pseudo adjoint optimization: harnessing the solution curve for SPICE acceleration. In: Xiong, J. and Wille, R., (eds.) ICCAD '24: Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design. ICCAD '24: 43rd IEEE/ACM International Conference on Computer-Aided Design, 27-31 Oct 2024, New York, USA. ACM ISBN 9798400710773

https://doi.org/10.1145/3676536.3676789

Pseudo Adjoint Optimization: Harnessing the Solution Curve for SPICE Acceleration

Anonymous Author(s)

ABSTRACT

Pseudo transient analysis (PTA) has been a promising solution for direct current (DC) analysis of transistor-level circuit simulation. Despite its popularity, PTA requires meticulous hyperparameter tuning for optimal performance. In this paper, we propose pseudo adjoint optimization, Soda-PTA, which models the PTA solution curve (which is used to measure convergence) using a neural ordinary differential equation (Neural ODE) and deriving explicit gradients of PTA iteration w.r.t. the PTA hyperparameters through the classic adjoint method, enabling effective optimization of the PTA hyperparameters. To generalize Soda-PTA for unseen circuits, we further introduce a graph convolution network to transfer optimal PTA hyperparameters from the other circuits to the target one. Soda-PTA is implemented in an out-of-the-box SPICE simulator. Through extensive experiments, Soda-PTA demonstrates superior acceleration performance: an average speedup of 1.53x over the state-of-the-art BoA-PTA while ensuring superior convergence and up to 22.12x speedup compared to the native PTA solver.

KEYWORDS

Circuit simulation, Nonlinear DC analysis, Pseudo transient analysis, Neural ODE, Graph convolution network

1 INTRODUCTION

Direct current (DC) analysis is a crucial step in SPICE simulation to determine the static operating points, which provides initial solutions for transient analysis and establishes small-signal parameters such as transconductance for semiconductor devices [5]. The DC analysis process involves solving a set of nonlinear algebraic equations generated from the circuit netlist using the Modified Nodal Analysis (MNA) method [7].

However, with the exponential growth in integration and complexity of integrated circuits driven by semiconductor technology advancements, the algebraic systems to be solved in DC analysis have become considerably larger and highly nonlinear. The basic Newton-Raphson (NR) [4] method is no longer suitable for these challenges. Consequently, various continuation methods for DC analysis have been extensively researched, including Gmin Stepping [14], Source Stepping [19], Homotopy [16], and Pseudo transient analysis (PTA) [12], among others. Unfortunately, Gmin Stepping and Source Stepping may fail to converge in situations involving large-scale circuits with strong non-linearity and discontinuity. Although Homotopy can ensure global convergence under certain conditions, its implementation heavily relies on device models, rendering it impractical in real simulators.

Among various continuation methods, PTA and its variants, including Pure PTA (PPTA) [6, 25], Damped PTA (DPTA) [26], Ramping PTA (RPTA) [11], and Compound Element PTA (CEPTA) [10], have proven to be the most promising approaches for solving large-scale and strongly nonlinear DC analysis problems, due to their ease of implementation and the absence of discontinuity issues [8]. The PTA algorithm transforms nonlinear algebraic systems into ordinary differential equaions (ODEs) by inserting pseudo-elements into the circuit [12]. Subsequently, numerical integration methods, i.e. Backward Euler [18], can be employed to solve this ODE system iteratively. At each discrete time point, a set of nonlinear equations is solved using the NR method.

The efficiency of solving the ODE systems is influenced by two aspects: time-step control and initial parameter selection. Regarding time-step control, there has been considerable prior research. Conventional PTA algorithms, even those employed in commercial EDA tools [15, 22], utilize a simple iteration counting approach to determine time-step size [20]. X. Wu et al. [27] proposed an adaptive time-step control method based on Switched Evolution/Relaxation (SER), which is a heuristic approach leveraging domain experiences. Z. Jin [9] and D. Niu [21] utilized advanced reinforcement learning to intelligently select the optimal time-step size for accelerating the solution of ODE systems. However, research on initial parameter selection is scarce. Unfortunately, for different circuits, the required pseudo-elements may vary significantly. For any given circuit, the problem of selecting optimal pseudo-elements i.e. PTA hyperparameters to minimize the total number of NR iteration (primary metric for evaluating PTA performance) does not have a definitive answer.

W. Xing et al. [29] introduced Bayesian Optimization Accelerated PTA (BoA-PTA), which focuses exclusively on the relationship between PTA hyperparameters and the total number of NR iterations. It employs a modified Gaussian process (GP) to characterize this relationship for updating the next iteration of PTA hyperparameters. Unfortunately, this approach appears to overlook the information embedded within the PTA iterative process and does not utilize judicious information in the evolution of the ODE system toward a steady state of the inserted pseudo-elements.

To this end, we present a novel solution-curve-based adjoint optimization to accelerate the PTA solver. Specifically, unlike conventional works where the solver is considered as a black box, we leverage the convergence information (in the solution curve) from the PTA during its simulation and model them using a Neural Ordinary Differential Equations (Neural ODE) [3]. We then define a novel loss function to measure the convergence and derive the optimization gradient for the PTA hyperparameters. For unseen circuits, we introduce a graph convolution network (GCN) to embed the design topology onto feature space, which allows effective knowledge transfer between different circuit designs and thus the optimal PTA hyperparameters for an unseen circuit. The contributions of this work are as follows:

- To the best of our knowledge, this is the first SPICE acceleration method by harnessing the convergence information collected while solving the system ODE, delivering an accurate hyperparameters optimization gradient explicitly.

• Soda-PTA is equipped with a GCN and allows knowledge transfer for different circuits and the acceleration can be generalized to unseen circuits.

• We demonstrate a significant speedup over the SOTA AI-accelerated SPICE BoA-PTA, with an average speedup of 1.53x and a maximum of 1.90x, while ensuring superior convergence. Moreover, extending Soda-PTA to PPTA, DPTA, and RPTA, the average acceleration ratios are 2.26x, 14.77x, and 22.12x respectively.

• Soda-PTA showcases a novel direction of AI implementation for standard EDA pipelines without introducing instability and errors because all interventions happen in the SPICE and the error can be monitored and controlled.

2 BACKGROUND

2.1 PTA for Nonlinear DC Simulation

Consider a nonlinear circuit with N nodes excluding the ground node, among which there are M independent voltage sources. Thus, finding the DC operating point is equivalent to solving the nonlinear system Eq. (1).

$$\mathbf{F}(\mathbf{x}) = 0 \tag{1}$$

where $\mathbf{x} = (v, i)^T \in R^n$, $n = N + M$, variable vector $v \in R^N$ denotes node voltage, and vector $i \in R^M$ represents internal branch currents of the independent voltage sources. The conventional PTA algorithm, PPTA, involves serially connecting a virtual inductor to each independent voltage source and each nonlinear voltage-related branch, and in parallel, connecting a virtual capacitor to each independent current source and each nonlinear current-related branch. Eq. (1) is transformed into a ODE system, denoted as Eq. (2).

$$\begin{cases} \mathbf{D}\dot{\mathbf{x}}(t) = -\mathbf{F}(\mathbf{x}(t), t) \\ \mathbf{x}(t_0) = \mathbf{x}_0 \end{cases} \tag{2}$$

where $\dot{\mathbf{x}}(t) = (\dot{v}(t), \dot{i}(t))$ is the derivative of $\mathbf{x}$ with respect to the time t, and $\mathbf{D}$ is the incidence matrix that represents the inserted pseudo-elements. Subsequently, the BE method is employed for transient analysis of this virtual circuit. Throughout this process, truncation error magnitude is disregarded, and the selection of integration time-step size is not bound by precision requirements, until reaching steady-state.

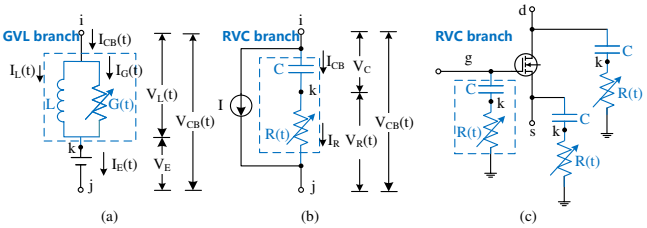

Figure 1: Inserted pseudo-elements and embedding positions.

In contrast to PPTA, CEPTA inserts a GVL branch into the independent voltage source in series, a RVC branch into the independent current source in parallel, and transistors between each node to ground [10], as shown in Figure 1. The relationships for time-variant resistor and time-variant conductor respectively are: $R(t) = R_0 e^{t/\tau}$ and $G(t) = G_0 e^{t/\tau}$, where R_0 and G_0 is a given initial value and τ is the time constant. Inserted GVL and RVC branch are stamped into the circuit matrix following the format outlined in Table 1, where G_{CBeq}, I_{CBeq}, R_{CBeq} and V_{CBeq} defined in Eq. (3) and Eq. (4). Therefore, we denote the CEPTA hyperparameters as: $\theta_{CEPTA} = [C, L, R_0, G_0]$, and obviously these inserted pseudo-elements have a significant impact on simulation convergence and efficiency.

DPTA, like PPTA, focuses on inductor L and capacitor C [26], $\theta_{DPTA} \backslash \theta_{PPTA} = [C, L]$. Meanwhile, RPTA solely concentrates on capacitor C, $\theta_{RPTA} = [C]$. For the internal of independent voltage sources, function control is employed to ensure the convergence of PTA [11]. In reality, the circuit matrix mappings of these three PTA algorithms also correspond to Table 1. Only the redefinition of Eq. (3) and (4) is required.

Table 1: The stampping for Branch RVC and GVL.

Branch		i	j	I_E	RHS
RVC	i	G_{CBeq}	$-G_{CBeq}$		$-I_{CBeq}$
	j	$-G_{CBeq}$	G_{CBeq}		I_{CBeq}
GVL	i			-1	
	j			-1	
	BR	-1	-1	$-R_{CBeq}$	V_{CBeq}

$$G_{CB_{eq}}^{-1} = h^{n+1}/C + R(t^{n+1}),\ I_{CB_{eq}} = G_{CB_{eq}}(I_{CB}^n R(t^n) - V_{CB}^n) \tag{3}$$

$$R_{CB_{eq}}^{-1} = h^{n+1}/L + G^{n+1},\ V_{CB_{eq}} = R_{CB_{eq}}(-I_{CB}^n + G^n(V_{CB}^n - E)) + E \tag{4}$$

2.2 Problem Formulation

Consider the PTA with netlist denoted as ξ and solver hyperparameters $\boldsymbol{\theta}$ as a function,

$$\left(NR\_iters; M; \{\mathbf{x}_t\}_{t=1}^M\right) = PTA(\xi, \boldsymbol{\theta}) \tag{5}$$

where $\{\mathbf{x}_t\}$ is the collection of the states at each instant t_n for Eq. (2), M and NR_iters are the total number of PTA steps and NR iterations respectively, required to obtain the transient solution $\{\mathbf{x}_t\}$ and they are the most critical performance metrics for the PTA.

In a PTA system, while we can achieve sensitivity to circuit performance by applying a joint method to the circuit ODEs, it is challenging to derive the convergence gradient within the standard framework. This limitation suggests the need for an alternative approach or an adaptation of the existing framework to effectively compute the gradient. As a remedy, end-to-end optimization uses a surrogate to learn the mapping of $NR\_iters = \eta(\xi, \boldsymbol{\theta})$, based on which optimization algorithms can be applied to find the optimal $\boldsymbol{\theta}^*$ [29]. What is neglected is that useful information $\{\mathbf{x}_t\}$ is discarded. Therefore, we propose to include the information $\{\mathbf{x}_t\}$ into the surrogate model, which is the key to our method to improve the optimization performance.

2.3 Neural Ordinary Differential Equations

A Neural ODE [3, 28] is a neural network that learns the derivative of the hidden states w.r.t. time

$$\dot{h}(t) = f_{\mathbf{w}}(\mathbf{h}(t), t) \tag{6}$$

where $\mathbf{h}(t) \in R^n$ is the state, $\dot{h}$ denotes the time derivative of h, $f_{\mathbf{w}} : R^n \to R^n$ is a neural network model parameterized by $\mathbf{w}$. Given an initial state h_0, the solution of Eq. (6) can be obtained

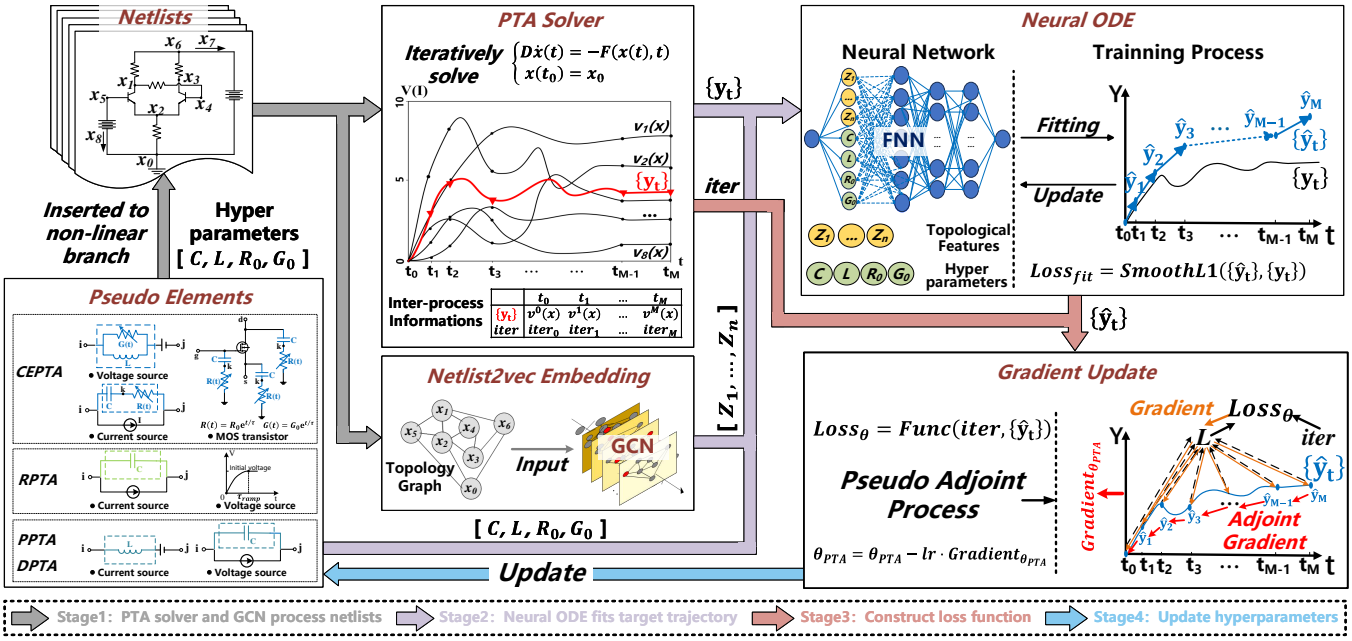

Figure 2: Entire flow of proposed Soda-PTA. ① PTA solver takes netlist input and outputs key state trajectory and the total number of NR iterations. GCN model embeds circuit design to a feature vector. ② Neural ODE uses the embedding features and PTA hyperparameters to fit the target trajectory, forming an effective surrogate for the PTA process. ③ Construct loss function with outputs from Neural ODE and PTA Solver, and get gradients of PTA hyperparameters. ④ Update PTA hyperparameters.

by any classic numerical solver for ODEs, e.g. Runge-Kutta (RK) methods and the solution is the dynamics of the states $\mathbf{h}(t)$. $f_{\mathbf{w}}$ can be expressed as: $f_{\mathbf{w}} = f_{\mathbf{w}_{1,2}} \circ \cdots \circ f_{\mathbf{w}_{k,k+1}}$, k is the number of layers, $f_{\mathbf{w}_{k,k+1}}$ is the forward process of the k-th layer, and $h_k = (f_{\mathbf{w}_{1,2}} \circ \cdots \circ f_{\mathbf{w}_{k,k+1}})(\cdot) \in R_{n_k}$ is the intermediate representation at the k-th layer, $h_K = \dot{h}$ is the output. $n_K = n$ denotes the number of neurons at k-th layer.

For the training of Neural ODE, the loss function is defined as:

$$L = \int_{t_0}^{t_1} l(\mathbf{h}(t), t)dt \tag{7}$$

where $l(\mathbf{h}(t), t)$ is the loss function (e.g., L2 loss) at time t. The gradient of the loss function w.r.t. the parameters $\mathbf{w}$, can be computed by the adjoint sensitivity method [3]:

$$\frac{\partial L}{\partial \mathbf{w}} = -\int_{t_1}^{t_0} a(t)^T \frac{\partial f_{\mathbf{w}}(\mathbf{h}(t), t)}{\partial \mathbf{w}} dt \tag{8}$$

where $a(t) = \frac{\partial L}{\partial \mathbf{h}(t)}$ is the adjoint, and $\mathbf{h}(t)$ is the intermediate state of Neural ODE at each instant. Its dynamics are defined by Eq. (8), where $\mathbf{h}(t)$ can simply recomputed backwards in time together with the adjoint.

3 PSEUDO ADJOINT OPTIMIZATION

3.1 Overall Framework

As discussed, what is missing in the literature is the inclusion of the process information (in this case, the whole solution curves), which leads to inferior performance. To this end, we propose to use the Neural ODE as a surrogate to imitate the PTA process. More importantly, once the Neural ODE is trained, the adjoint state of the Neural ODE can be used to compute the gradient of the loss function w.r.t. the PTA hyperparameters, which is not available in the original PTA solver.

The overall workflow of Soda-PTA is shown in Figure 2. Based on a given netlist and hyperparameters, PTA solves the system ODEs iteratively until the states $\{\mathbf{x}_t\}$ converge, based on which we summarize the dynamic of $\{\mathbf{x}_t\}$ using key trajectory $\{\mathbf{y}_t\}$. A Neural ODE is then introduced to fit $\{\mathbf{y}_t\}$ by minimizing the fitting loss. Once the fitting is completed, another loss of measuring the convergence is defined to enable the maximization of the convergence speed w.r.t. the PTA hyperparameters.

3.2 Solution Curve Modeling Using Neural ODE

We first model the solution curves of the node voltages as a whole dynamical system, encapsulating essential information about the PTA hyperparameters, objective optimization metrics, and circuit topology. Each node within PTA is inherently linked with a unique solution curve. However, modeling every solution curve is both unnecessary and impractical. Our primary aim is to minimize *NR_iters*, which can be effectively achieved by focusing on a subset of the key solution curves that accurately reflect the convergence of the PTA process.

In this pursuit, we propose using the first n-th moments of these solution curves as the target fitting trajectory $\mathbf{y}_t$. Our investigations reveal that the first moment, or the mean of the solution curves, suffices in modeling the convergence of the PTA process. Utilizing these solution curves, we apply Neural ODE to simulate the PTA process, as illustrated in the following equation:

$$\frac{d\mathbf{y}}{dt} = f_{\mathbf{w}}(\mathbf{y}(t), t, \boldsymbol{\theta}) \tag{9}$$

Here, $\mathbf{y}$ represents the discretized solution trajectory, $\mathbf{w}$ signifies the Neural ODE parameters, and $\boldsymbol{\theta}$ denotes the PTA hyperparameters.

The objective of the Neural ODE is to dynamically replicate the behavior of the PTA process.

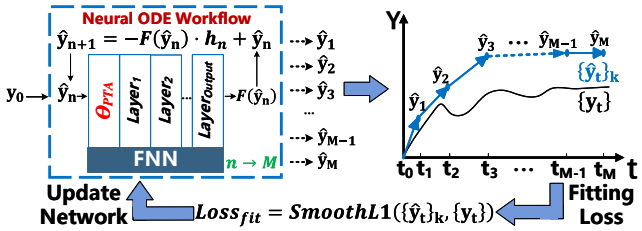

Figure 3: Neural ODE fits the target solution curve in the k-th fitting epoch.

A pivotal aspect of our methodology is the optimization process, which uses a Smooth L1 loss function, $l_{fit}(\cdot)$, to compare the predictive solution curve with the target, as shown in Figure 3. The gradient of this loss function w.r.t. Neural ODE parameters $\mathbf{w}$ yields the adjoint state $a(t)$, which in turn is employed to update the PTA hyperparameters (with details in Section 3.3 and Algorithm 1).

Despite efforts in standardizing the data for the target trajectory, striking a balance between fitting efficiency and learning rate scheduling remains challenging. To mitigate this, we have implemented an early stopping strategy based on the relative change in the fitting loss. Specifically, if $\frac{abs(l_{fit}^{k} - l_{fit}^{k-1})}{l_{fit}^{k}} < 0.001$ consistently over a fixed number of iterations, such as 10, we terminate the fitting process. This early termination often indicates either suboptimal PTA hyperparameter selection or complex convergence challenges in the PTA process due to specific circuit characteristics. Notably, terminating the fitting process does not imply an end to the optimization of PTA parameters. The formulation and impact of this strategy on PTA hyperparameter optimization are further elaborated in Section 3.3.

Moreover, appropriately reducing the length of $\mathbf{y}_t$ is also a necessary operation. For PTA processes that exhibit oscillatory non-convergence and prolonged convergence times, the target trajectory often comprises a significant number of PTA steps, which imposes a substantial overhead on fitting Neural ODE. Fortunately, we draw inspiration from another non-convergence scenario in the PTA process, "time-step too small", by selectively utilizing partial trajectory to accommodate unacceptable trajectory lengths. To this end, we employ an adaptive approach where, upon exceeding a certain threshold of PTA steps, only the first 500 PTA steps are retained. In section 4.4, analysis in dealing with non-convergence scenarios validates the effectiveness of this approach.

3.3 Pseudo Ajoint Optimization

Upon fitting the Neural ODE to the critical target trajectory, our focus shifts to deriving the convergence as a function of the PTA hyperparameters, formulated through a specialized loss function. This function not only reflects the deviation from the target trajectory but also incorporates the dynamics of the circuit analysis, as evidenced in the empirical correlation between the number of PTA execution steps (*PTA_steps*) and NR iterations (*NR_iters*), highlighted in Figure 4.

We introduce a novel loss function Eq. (10) that combines the count of NR iteration (*NR_iters*), states ($\|\hat{\mathbf{y}}_t\|_1$) and the cumulative magnitude ($\|(\hat{\mathbf{y}}_t - \hat{\mathbf{y}}_{t-1})\|_1$) of the generated trajectory, which is

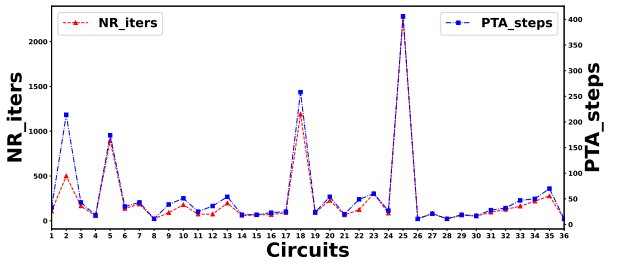

Figure 4: The relationship between NR_iters and PTA_steps for 36 circuits from benchmark [2] under default CEPTA hyperparameters.

inspired by Figure 5:

$$loss_{\theta}(\{\hat{\mathbf{y}}_t\}) = NR\_iters \cdot (\|\hat{\mathbf{y}}_t\|_1 + \|(\hat{\mathbf{y}}_t - \hat{\mathbf{y}}_{t-1})\|_1) \quad (10)$$

where $\hat{\mathbf{y}}_t$ represents the states on the trajectory generated by the Neural ODE. This loss function is pivotal in guiding the optimization of the PTA hyperparameters $\boldsymbol{\theta}$. Notably, its design, involving numerical computations along the trajectory, facilitates the updated $\boldsymbol{\theta}$ to progress towards improved values, even in cases of early stopping. This approach potentially offsets the costs associated with early termination by steering the optimization in a beneficial direction.

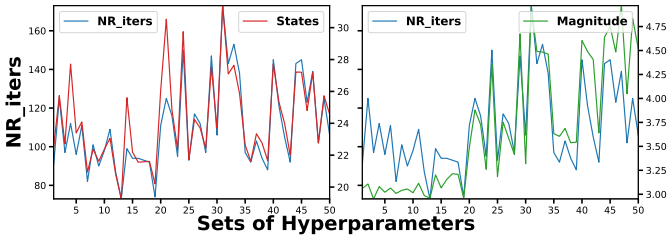

Figure 5: The relationship between NR_iters and two subcomponents of the loss function Eq. (10) for the "hussamp" circuit across 50 different sets of CEPTA hyperparameters.

Gradient calculations for updating $\boldsymbol{\theta}$ are expressed in Eq. (11) and Eq. (12), which cumulatively accounts for the contribution of each state along the trajectory:

$$\nabla\boldsymbol{\theta} = \sum_{t=1}^{M} \frac{\partial loss_{\theta}}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \boldsymbol{\theta}} \quad (11)$$

$$\boldsymbol{\theta} = \boldsymbol{\theta} - lr \cdot \nabla\boldsymbol{\theta} \quad (12)$$

where $\partial loss_{\theta}/\partial \hat{\mathbf{y}}_t$ corresponds to the adjoint state within the Neural ODE framework. Employing this methodology, we aim to minimize NR iterations by iteratively updating the PTA hyperparameters $\boldsymbol{\theta}$ in the direction indicated by the gradient in Eq. (11). Since these gradients are not the actual gradients of the SPICE solver but offer a viable direction for updating $\boldsymbol{\theta}$, we refer to this method as Pseudo Adjoint Optimization, with its details summarized in Algorithm 1.

3.4 Netlist2vec Embedding Through GCN

From Algorithm 1, we can see that the optimization of $\boldsymbol{\theta}$ for a given netlist is always conducted from scratch, despite that previous optimization for a similar circuit (e.g., a two-stage operational amplifier and a three-stage one). If we can learn the connection between circuits, we will be able to directly propose near-optimal

Algorithm 1 Soda-PTA Algorithm Framework

Input: PTA solver, Neural ODE $f_{\mathbf{w}}(\cdot)$, number of epoch N_{epoch}, default PTA hyperparameters $\boldsymbol{\theta}_0$

$\boldsymbol{\theta} = \boldsymbol{\theta}_0$
for $i = 1 \rightarrow N_{epoch}$ **do**
  $\left(NR\_iters; M; \{\mathbf{x}_t\}_{t=1}^{M}\right) = PTA(\xi, \boldsymbol{\theta})$
  Summarize $\{\mathbf{x}_t\}_{t=1}^{M}$ into $\{\mathbf{y}_t\}_{t=1}^{M}$
  $\mathbf{w} \leftarrow \text{argmin}_{\mathbf{w}} \left(l_{fit}(\{\hat{\mathbf{y}}_t\}, \{\mathbf{y}_t\})\right)$ using adjoint method
  $\boldsymbol{\theta} \leftarrow \text{argmin}_{\boldsymbol{\theta}} \left(l_{\boldsymbol{\theta}}(\{\hat{\mathbf{y}}_t\})\right)$ using adjoint method
end for
return $\boldsymbol{\theta}$

hyperparameters given a new circuit, which will be refined through iterations. The emerging AI technique, GCN, has been extensively applied in the EDA domain [23, 24], notably in areas such as reliability and security in integrated circuit design [1], as well as in the realm of testability analysis [17].

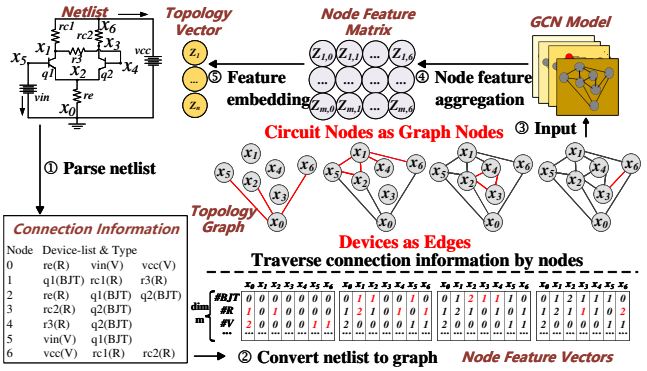

Figure 6: Netlist2vec embedding process.

To this end, we utilize GCN to characterize the netlist, fostering expedited optimization and superior optimization outcomes [13]. In contrast to conventional methodologies that rely on simple circuit summary factors [29], our approach capitalizes on more extensive automatic feature extraction by incorporating circuit topology information. Drawing inspiration from the MNA matrix formulation, the netlist is mapped here as a graph structure with circuit nodes as vertices and devices as edges. The specific process is shown in Figure 6. We further embed the vector of circuit features ξ for the subsequent learning of Neural ODE.

$$\frac{d\mathbf{y}}{dt} = f_{\mathbf{w}}(\mathbf{y}(t), t, \boldsymbol{\theta}, \xi) \tag{13}$$

Substituting circuit feature vector ξ into the Neural ODE, our model becomes Eq. (13), which is fitted using previously described fitting loss function $l_{fit}(\cdot)$ and the optimization is conducted exactly as in Eq. (11) and Eq. (12) with the only difference being that optimization is conditional on a given ξ. Given an unseen circuit, our model will construct a feature vector ξ and then guess the solver trajectory, based on which the optimal PTA hyperparameters $\boldsymbol{\theta}$ are obtained. The specific steps are illustrated in Algorithm 2. Certainly, this guess might not be accurate, but it is a good starting point for the optimization process. Also, if this model is trained on a large number of circuits, it will be able to generalize to unseen circuits as we will see in the experiments.

Algorithm 2 Soda-PTA for unseen circuit with GCN

Input: Algorithm 1, $GCN(\cdot)$, test set Te, train set Tr, test iteration N_{epoch}^{te}, train iteration N_{epoch}^{tr}

Initilize Neural ODE $f_{\mathbf{w}}(\cdot)$
for μ in Tr **do**
  $\xi = GCN(\mu)$
  $\boldsymbol{\theta}^*(\xi) = Algorithm1(f_{\mathbf{w}}(\cdot), \boldsymbol{\theta}_0, \xi, N_{epoch}^{tr})$
end for
Update GCN by the total loss of all generated trajectories
for μ in Te **do**
  $\xi = GCN(\mu)$
  $\boldsymbol{\theta}^* = Algorithm1(f_{\mathbf{w}}(\cdot), \boldsymbol{\theta}_0, \xi, N_{epoch}^{te})$
end for

4 EXPERIMENTS AND RESULTS

4.1 Experimental Setup

We implement the proposed Soda-PTA in a SPICE-like circuit simulator on AMD 4900H CPU, and train neural networks on NVIDIA GeForce RTX 2060 6GB GPU. The hyperparameters to be optimized and their default values for various PTA algorithms are shown in Table 2. For a comprehensive assessment, Soda-PTA is evaluated on a canonical benchmark [2], collected challenging test cases, and multiple real-world complex problems.

Our primary performance metric of interest is the solution time, indicated by NR_iters. The SOTA work BoA-PTA [29] is conducted under three acquisition functions: UCB, MES, and EI, to obtain optimal results. Unless stated otherwise, all PTA uses a simple iteration counting time-step control [22] for a fair comparison.

Table 2: The hyperparameters and their default values of various PTA algorithms.

PTA algorithms	PPTA	DPTA	RPTA	CEPTA
Hyperparameters	C, L	C, L	C	C, L, R_0, G_0
Default values	[1e-4, 1]	[1e-4, 1]	[1e-4]	[1e-4, 1, 100, 1e-5]

In this work, Soda-PTA is utilized to dynamically select optimal PTA hyperparameters online to enhance the performance and convergence of nonlinear DC simulation. Section 4.2 demonstrates the fitting effectiveness of Neural ODE. In Section 4.3, we compare Soda-PTA with the SOTA BoA-PTA solver [29] under CEPTA, while also extending Soda-PTA to other PTA algorithms, illustrating the effectiveness of Soda-PTA. Section 4.4 analyzes two scenarios of non-convergence and validates the robustness of Soda-PTA. Section 4.5 introduces GCN to further improve the optimization quality of Soda-PTA. Finally, in section 4.6, we apply Soda-PTA to RL-S [9], which leverages reinforcement learning to intelligently select time-step, thereby accelerating PTA.

4.2 Solution Curve Modeling Performance

Figure 7 and Figure 8 represent the neural ODE fitting process for two circuits under CEPTA and DPTA, respectively. As mentioned

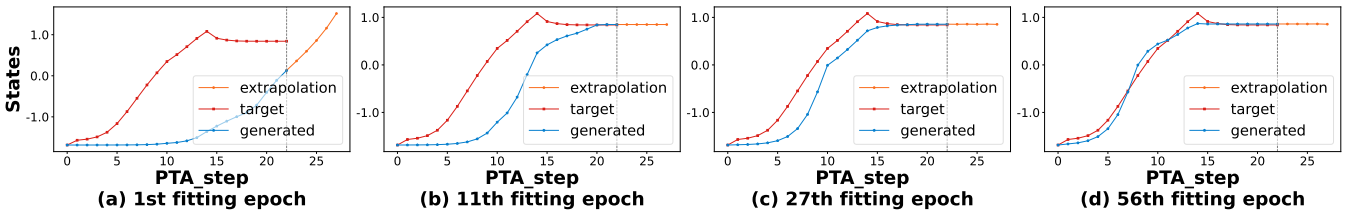

Figure 7: Fitting process of Neural ODE for the "hussamp" circuit under CEPTA.

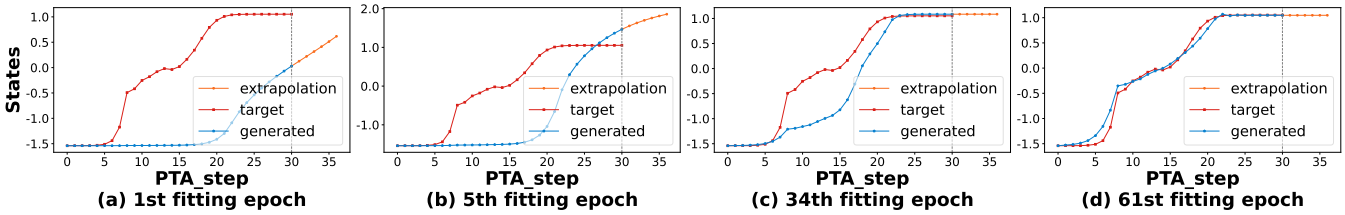

Figure 8: Fitting process of Neural ODE for the "6stageLimAmp" circuit under DPTA.

above, we use the mean of the solution curves as the key target trajectory.

The red line represents the target key trajectory obtained from SPICE simulation, while the blue line represents the learned trajectory from the fitting training data, and the orange additional portion of the blue line corresponds to our subsequent predictive outputs. Specifically, after the fitting training, Neural ODE continues to produce subsequent data points using a fixed time-step, identical to the last time-step of the generated trajectory. Notably, as the fitting training process progresses, Neural ODE gradually approaches the target trajectory, enabling the model to robustly achieve convergence.

4.3 PTA Acceleration Comparisons

To comprehensively evaluate the optimization performance of the proposed Soda-PTA, we have tested it under four PTA algorithms (PPTA [25], DPTA [26], CEPTA [10] and RPTA [11]), and compared it with the native PTA and BOA-PTA (which is a SOTA enhance of PTA). Convergence and simulation efficiency will be comparatively analyzed.

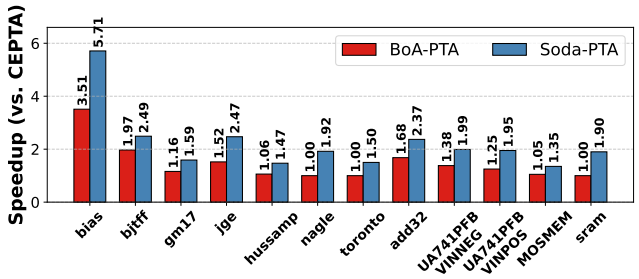

Figure 9: Simulation performance comparison under CEPTA.

Firstly, we compare Soda-PTA and BoA-PTA across 12 test circuits, as depicted in Figure 9. It can be observed that Soda-PTA exhibits an average improvement of 1.53x, with a maximum improvement of 1.9x compared to BoA-PTA. Subsequently, we select four different types of circuits from the benchmark to compare the optimization processes of Soda-PTA and BoA-PTA, primarily reflecting their utilization of SPICE resources. Figure 10 depicts the comparison of the first 10 update epochs of PTA hyperparameter optimization. It can be seen that Soda-PTA requires fewer optimization epochs to achieve comparable optimization performance compared to BoA-PTA, indicating less utilization of SPICE resources.

We then extend Soda-PTA to other PTA algorithms and analyze the optimization effects of Soda-PTA across 20 test circuits, as shown in Table 3. From the table, it is evident that Soda-PTA exhibits significant acceleration, with average speedups of 2.11x and maximum speedups of 5.71x under CEPTA and maximum speedups of 4.74x under PPTA. The acceleration is particularly pronounced for DPTA and RPTA, with average speedups of 14.77x and 22.12x, respectively. Notably, for the circuits "ab_opamp", "schmitfast", "MOSMEM" and "UA709", a substantial reduction in the number of NR iteration is observed. This can be attributed to the core concept of Soda-PTA, which leverages inter-process trajectory information to learn surrogate models, effectively guiding the convergence of PTA process and the formation of PTA hyperparameter gradients. Consequently, this significantly reduces the number of NR iterations.

Additionally, under PPTA, Soda-PTA cannot guarantee convergence for some circuits due to instances of "time-step too small" non-convergence phenomena occurring in the early stages of the PTA process. This results in trajectory sequences being too short to enable the Neural ODE to capture critical information of the PTA process effectively and form a valid surrogate to guide the formation of effective gradient information for PTA hyperparameters. Although for the "Multiplier" circuit under CEPTA, similarly experiencing "time-step too small" non-convergence issues, it provides sufficient trajectory information for Soda-PTA to handle effectively. Furthermore, for the circuits "bias", "nand", and "MOSAMP1" non-convergence stemming from oscillations is observed, and Soda-PTA effectively ensures convergence in these cases.

It is essential to conduct testing across other PTA algorithms. While CEPTA significantly mitigates oscillation issues in the traditional PTA algorithms, it is not always optimal. The numbers highlighted in the "Soda-PTA " column in the Table 3 represent the

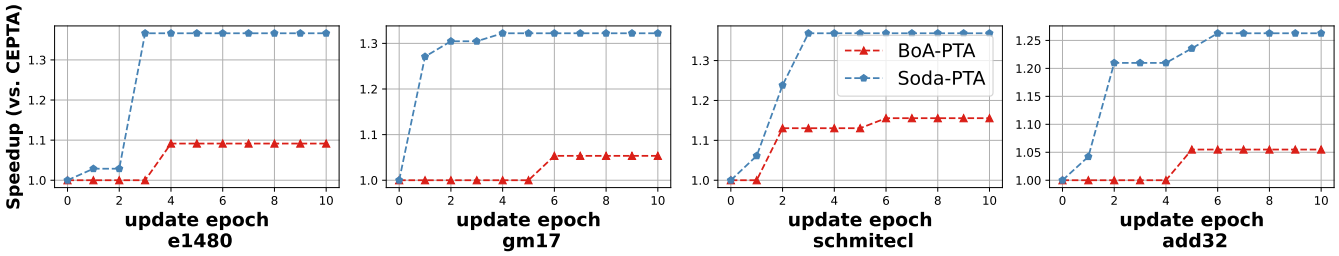

Figure 10: Optimization process comparison between Soda-PTA and BoA-PTA.

Table 3: Simulation performance comparison under CEPTA, PPTA, DPTA and RPTA. "—" denotes non-convergence. The numbers highlighted represent the best results among these PTA algorithms.

circuits	NR_iters								Speedup			
	CEPTA		PPTA		DPTA		RPTA					
	native	Soda-PTA	native	Soda-PTA	native	Soda-PTA	native	Soda-PTA	vs. CEPTA	vs. PPTA	vs. DPTA	vs. RPTA
ab_opamp	150	***110***	—	—	2417	146	2408	127	1.36x	—	16.55x	18.96x
astabl	55	45	108	64	81	43	75	***41***	1.22x	1.69x	1.88x	1.83x
bias	839	147	—	899	755	607	498	***110***	5.71x	—	1.24x	4.53x
bjtinv	186	53	125	77	155	***51***	101	101	3.51x	1.62x	3.04x	1.00x
cram	91	88	—	—	130	100	128	***81***	1.03x	—	1.30x	1.58x
gm6	69	42	—	—	110	55	107	***38***	1.64x	—	2.00x	2.82x
hussamp	91	***62***	—	—	209	87	240	71	1.47x	—	2.40x	3.38x
mosrect	65	***51***	251	53	838	63	837	55	1.27x	4.74x	13.30x	15.22x
nand	83	53	—	***32***	—	142	—	76x	1.57x	—	—	—
schmitfast	82	59	71	***30***	5681	106	5678	92	1.39x	2.37x	53.59x	61.72x
6stageLimAmp	137	51	69	***38***	135	73	137	51	2.69x	1.82x	1.85x	2.69x
add32	173	73	—	—	1765	234	1970	***70***	2.37x	—	7.54x	28.14x
DCOSC	126	***78***	108	91	116	98	136	100	1.62x	1.19x	1.18x	1.36x
DIFFPAIR	148	57	101	71	114	109	137	***47***	2.60x	1.42x	1.05x	2.91x
MOSAMP1	122	82	—	139	158	96	162	***69***	1.49x	—	1.65x	2.35x
MOSBandgap	153	***85***	—	—	342	113	341	104	1.80x	—	3.03x	3.28x
MOSMEM	127	***94***	253	98	26029	171	26037	101	1.35x	2.58x	152.22x	257.79x
TADEGLOW	103	63	151	***51***	164	66	86	60	1.63x	2.96x	2.48x	1.43x
UA709	407	***110***	311	143	2985	219	3270	887	3.70x	2.17x	13.63x	3.69x
Multiplier	—	105	—	—	232	***92***	225	94	—	—	2.52x	2.39x
Average									2.11x	2.26x	14.77x	22.12x

best results in parameter optimization among these four PTA algorithms. It is noteworthy that these results are consistently better than the native CEPTA, and it is evident that the optimized effects of Soda-PTA correspond to different PTA algorithms. This provides a crucial insight that, when dealing with unseen circuits and lacking an optimal decision on which PTA algorithm to execute, applying parameter optimization strategy Soda-PTA can consistently achieve higher performance.

4.4 PTA Convergence Comparisons

In addition to simulation performance, PTA convergence assurance is actually more crucial and promising, especially for large-scale circuits and circuits with convergence challenges. As shown in Table 4, we list some circuits under three commonly used PTA algorithms, all of which are difficult to converge under default hyperparameters. From the table, it is apparent that BoA-PTA provides weaker convergence assurance compared to proposed Soda-PTA.

Figure 11 illustrates two non-convergence scenarios under the PTA algorithms. We elucidate the "time-step too small" issue using the "opampal" circuit from Table 4 and the non-convergence issue caused by oscillation using the "Divider" circuit. The vertical axis of Figure 11 represents the voltage value of a specific node during the PTA process, while the horizontal axis represents discrete time points during the PTA process. For Figure 11.(a), it can be observed

Table 4: Conergence analysis. "—" denotes non-convergence.

PTA Algorithms	Circuits	NR_iters		
		native	BoA-PTA	Soda-PTA
CEPTA	opampal	— (time-step too small)	635	317
	D10		65	60
	loc	— (oscillation)	—	328
	ram2k		188	158
DPTA	gm17	— (oscillation)	N/A	304
	gm19		N/A	160
	REGULATOR		N/A	644
	Divider		N/A	511
RPTA	Schmitslow	— (oscillation)	N/A	4507
	bjtff		N/A	1458
	toronto		N/A	1484
	sram		N/A	2341

from the left subplot that at the final time-step, the numerical integration time-step size falls below the lowest limit (1e-9), resulting in non-convergence of the PTA process. However, the sufficient inter-process information it provides can be leveraged by Soda-PTA for learning, guiding the gradient updates of PTA parameters, ultimately leading to the convergence scenario depicted in the right subplot. As for Figure 11.(b), the left subplot clearly exhibits oscillatory behavior, while the right subplot demonstrates the results after parameter optimization.

Notably, the oscillatory behavior depicted in the left subplot of Figure 11.(b) can lead to prolonged non-convergence issues, resulting in the generation of a significant amount of inter-process information in the PTA process, posing a considerable challenge to the learning cost of Neural ODE. Fortunately, the core idea of Soda-PTA is to surrogate the PTA process by learning the target trajectory, which prompts us to adaptively truncate long sequences to balance the learning cost with optimization effectiveness. As demonstrated in Table 4, our strategy proves to be effective.

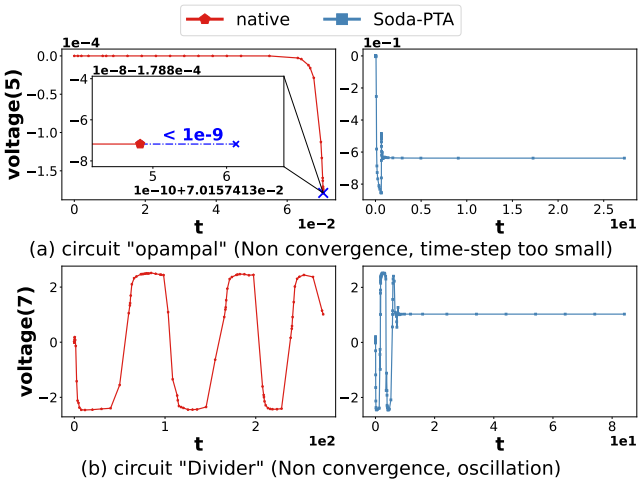

Figure 11: Two Non-convergence scenarios.

4.5 Unseen Circuit Performance with GCN

GCN extracts circuit topological features in order to reduce the additional calculations caused by executing the solver multiple times when finding the optimal parameters for the PTA solver corresponding to a given circuit. Therefore, we use various practical circuits outside the training set (bennchmark [2]) to test the performance of Soda-PTA combined with GCN models under CEPTA and DPTA, as shown in Figure 12.

The first subplot of Figure 12 depicts tests conducted under CEPTA, where from circuits "D20" and "D21", it is evident that the introduction of GCN model leads to an improvement in optimization results while reducing the required parameter update iterations. Additionally, for the "TRACKTorig" circuit, under the condition of maintaining optimization results almost unchanged, there is a significant reduction in the required update iterations. Similar conclusions are observable under DPTA as well. The circuit "D10", despite undergoing more update iterations, demonstrates more pronounced improvements during the optimization process.

In summary, it can be affirmed that GCN provides a stable mapping relationship from circuit descriptions to vector spaces, thereby cooperatively participating in the parameter optimization process and ensuring the quality of the optimization process.

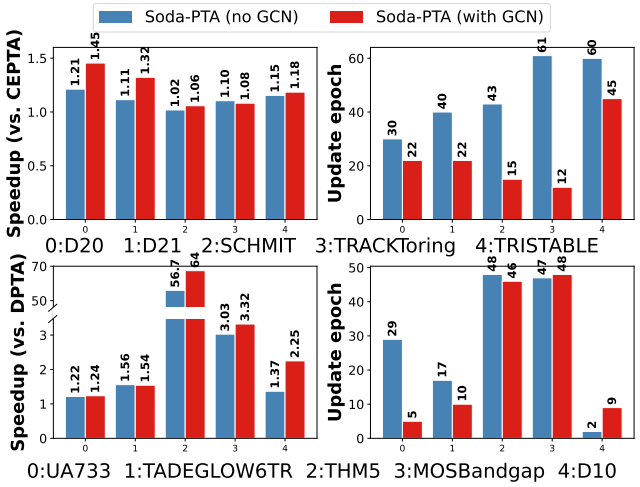

Figure 12: Improvement effect of GCN on Soda-PTA under CEPTA and DPTA.

4.6 Comparisons with PTA Algorithm Utilizing Advanced Time-Step Strategy

To assess Soda-PTA with more advanced time-step control, we apply RL-S [9], which employs reinforcement learning to select time steps, to Soda-PTA. The results are illustrated in Figure 13. Compared to the results without initial parameter selection, Soda-PTA achieves an average improvement of 1.53x in terms of *NR_iters*, with a maximum improvement of 7.55x. In terms of *PTA_steps*, Soda-PTA achieves an average improvement of 1.46x, with a maximum improvement of 5.76x.

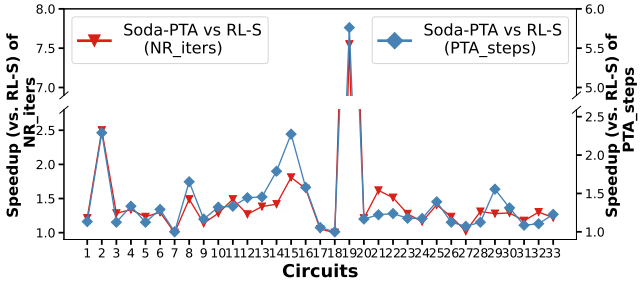

Figure 13: Speedup of Soda-PTA over RL-S.

5 CONCLUSION

In this paper, we propose a novel pseudo adjoint framework to optimize the hyperparameters in various PTA algorithms. This framework is further enhanced by the incorporation of GCN to expedite the process of transfer learning to deal with unseen circuits. Evaluated on benchmark circuits, Soda-PTA outperforms native CEPTA, PPTA, DPTA and RPTA by 2.11x, 2.26x, 14.77x and 22.12x respectively. Moreover, Soda-PTA exhibits superior acceleration performance and enhanced convergence capabilities compared to BoA-PTA. Applied to RL-S employing advanced time-step control strategy, the initial PTA hyperparameters provided by Soda-PTA result in an average acceleration of 1.53x.

REFERENCES

[1] Lilas Alrahis, Johann Knechtel, and Ozgur Sinanoglu. 2023. Graph Neural Networks: A Powerful and Versatile Tool for Advancing Design, Reliability, and Security of ICs. In *ASPDAC*. https://doi.org/10.1145/3566097.3568345

[2] J.A. Barby and R. Guindi. 1993. CircuitSim93: A circuit simulator benchmarking methodology case study. In *Sixth Annual IEEE International ASIC Conference and Exhibit*. https://doi.org/10.1109/ASIC.1993.410775

[3] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. 2018. Neural Ordinary Differential Equations. In *NIPS*.

[4] Leon O Chua. 1975. Computer-aided analysis of electronic circuits. *Algorithms and computational techniques* (1975).

[5] J. Deng, K. Batselier, Y. Zhang, and N. Wong. 2014. An efficient two-level DC operating points finder for transistor circuits. In *DAC*. https://doi.org/10.1145/2593069.2593087

[6] L. Goldgeisser, E. Christen, M. Vlach, and J. Langenwalter. 2001. Open ended dynamic ramping simulation of multi-discipline systems. In *ISCAS*. https://doi.org/10.1109/ISCAS.2001.922046

[7] Chung-Wen Ho, Albert Ruehli, and Pierce Brennan. 1975. The modified nodal approach to network analysis. *IEEE Transactions on circuits and systems* (1975).

[8] Z. Jin, M. Liu, and X. Wu. 2018. An adaptive dynamic-element PTA method for solving nonlinear DC operating point of transistor circuits. In *MWSCAS*. https://doi.org/10.1109/MWSCAS.2018.8623955

[9] Z. Jin, H. Pei, Y. Dong, X. Jin, X. Wu, W. W. Xing, and D. Niu. 2022. Accelerating Nonlinear DC Circuit Simulation with Reinforcement Learning. In *DAC*. https://doi.org/10.1145/3489517.3530512

[10] Zhou Jin, Xiao Wu, and Yasuaki Inoue. 2013. An effective implementation and embedding algorithm of PTA method for finding DC operating points. In *ICCCAS*. https://doi.org/10.1109/ICCCAS.2013.6765265

[11] Zhou Jin, Xiao Wu, Yasuaki Inoue, and Niu Dan. 2014. A ramping method combined with the damped PTA algorithm to find the DC operating points for nonlinear circuits. In *ISIC*. https://doi.org/10.1109/ISICIR.2014.7029506

[12] C. T. Kelley and D. E. Keyes. 1998. Convergence analysis of pseudo-transient continuation. *SIAM J. Numer. Anal.* (1998).

[13] Thomas Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. *ArXiv* (2016). https://api.semanticscholar.org/CorpusID:3144218

[14] K. S. Kundert and P. Gray. 1995. *The Designer's Guide to Spice and Spectre*. Kluwer Academic Publishers.

[15] K. S. Kundert and P. C. Gray. 1995. *The designer's guide to spice and spectre*.

[16] C. E. Lemke. 1984. Pathways to Solutions, Fixed Points, and Equilibria (C. B. Garcia and W. J. Zangwill). *SIAM Rev.* (1984). https://doi.org/10.1137/1026093 arXiv:https://doi.org/10.1137/1026093

[17] Yuzhe Ma, Haoxing Ren, Brucek Khailany, Harbinder Sikka, Lijuan Luo, Karthikeyan Natarajan, and Bei Yu. 2019. High Performance Graph Convolutional Networks with Applications in Testability Analysis. In *DAC*. https://doi.org/10.1145/3316781.3317838

[18] Benoıt Merlet and Morgan Pierre. 2009. Convergence to equilibrium for the backward Euler scheme and applications. *Convergence* (2009).

[19] TM Najibi. 1989. Continuation methods as applied to circuit simulation. *IEEE Circuits and Devices Magazine* (1989).

[20] F. N. Najm. 2010. *Circuit simulation*. John Wiley Sons.

[21] Dan Niu, Yichao Dong, Zhou Jin, Chuan Zhang, Qi Li, and Changyin Sun. 2023. OSSP-PTA: An Online Stochastic Stepping Policy for PTA on Reinforcement Learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2023). https://doi.org/10.1109/TCAD.2023.3251731

[22] H. R. Pota. 2010. *Inside spice*.

[23] Haoxing Ren, Siddhartha Nath, Yanqing Zhang, Hao Chen, and Mingjie Liu. 2022. Why are Graph Neural Networks Effective for EDA Problems? (Invited Paper). In *ICCAD*. Article 1, 8 pages. https://doi.org/10.1145/3508352.3561093

[24] Daniela Sánchez, Lorenzo Servadei, Gamze Naz Kiprit, Robert Wille, and Wolfgang Ecker. 2023. A Comprehensive Survey on Electronic Design Automation and Graph Neural Networks: Theory and Applications. *ACM Trans. Des. Autom. Electron. Syst.*, Article 15 (feb 2023). https://doi.org/10.1145/3543853

[25] W. Weeks, A. Jimenez, G. Mahoney, D. Mehta, H. Qassemzadeh, and T. Scott. 1973. Algorithms for ASTAP–A network-analysis program. *IEEE Transactions on Circuit Theory* (1973). https://doi.org/10.1109/TCT.1973.1083755

[26] Xiao Wu, Zhou Jin, and Yasuaki Inoue. 2013. Numerical integration algorithms with artificial damping for the PTA method applied to DC analysis of nonlinear circuits. In *ICCCAS*. https://doi.org/10.1109/ICCCAS.2013.6765266

[27] Xiao Wu, Zhou Jin, Dan Niu, and Yasuaki Inoue. 2017. An Adaptive Time-Step Control Method in Damped Pseudo-Transient Analysis for Solving Nonlinear DC Circuit Equations. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* (02 2017). https://doi.org/10.1587/transfun.E100.A.619

[28] W. Xiao, T. Wang, R. Hasani, M. Lechner, Y. Ban, C. Gan, and D. Rus. 2023. On the Forward Invariance of Neural ODEs. In *ICML*.

[29] W. W. Xing, X. Jin, T. Feng, D. Niu, W. Zhao, and Z. Jin. 2022. BoA-PTA: A Bayesian Optimization Accelerated PTA Solver for SPICE Simulation. *ACM TRANSACTIONS ON DESIGN AUTOMATION OF ELECTRONIC SYSTEMS* (2022). https://doi.org/10.1145/3555805