Concurrent Sign-off Timing Optimization via Deep Steiner Points Refinement

Siting Liu[1,2†], Ziyi Wang[1†], Fangzhou Liu[1], Yibo Lin[2], Bei Yu[1], Martin Wong[1]

[1]Chinese University of Hong Kong, [2]Peking University

Introduction

- Optimizing the sign-off metrics in early stages requires a full time-consuming physical design flow.
- Recent progress in machine learning (ML) has permitted fast and precise evaluation skipping a complex process.
- Early-stage timing optimization is critical for timing closure to reduce the turnaround time.

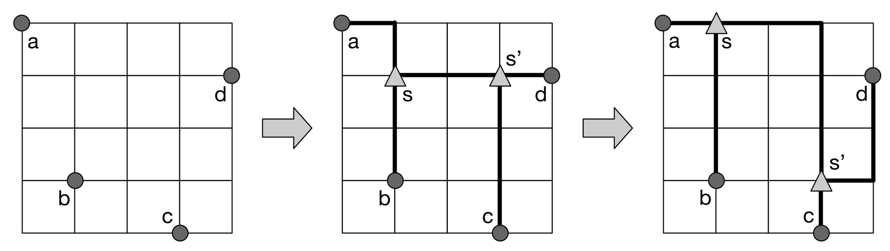
Background

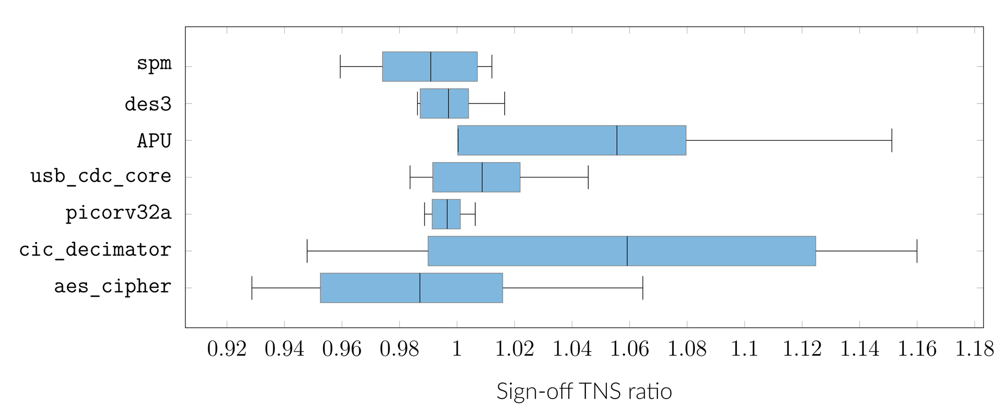
Timing Closure

- Slack: $s_e = r_e - a_e$ for a timing path endpoint e, where r_e and a_e denote e's required time and arrival time.
- Worst Negative Slack (WNS): $w(\cdot) = \min_e s_e$.
- Total Negative Slack (TNS): $t(\cdot) = \sum_e \{\min\{0, s_e\}\}$

Timing awareness has been extended to most phases of the physical design flow for the timing closure.

Timing Optimization via Steiner point Refinement



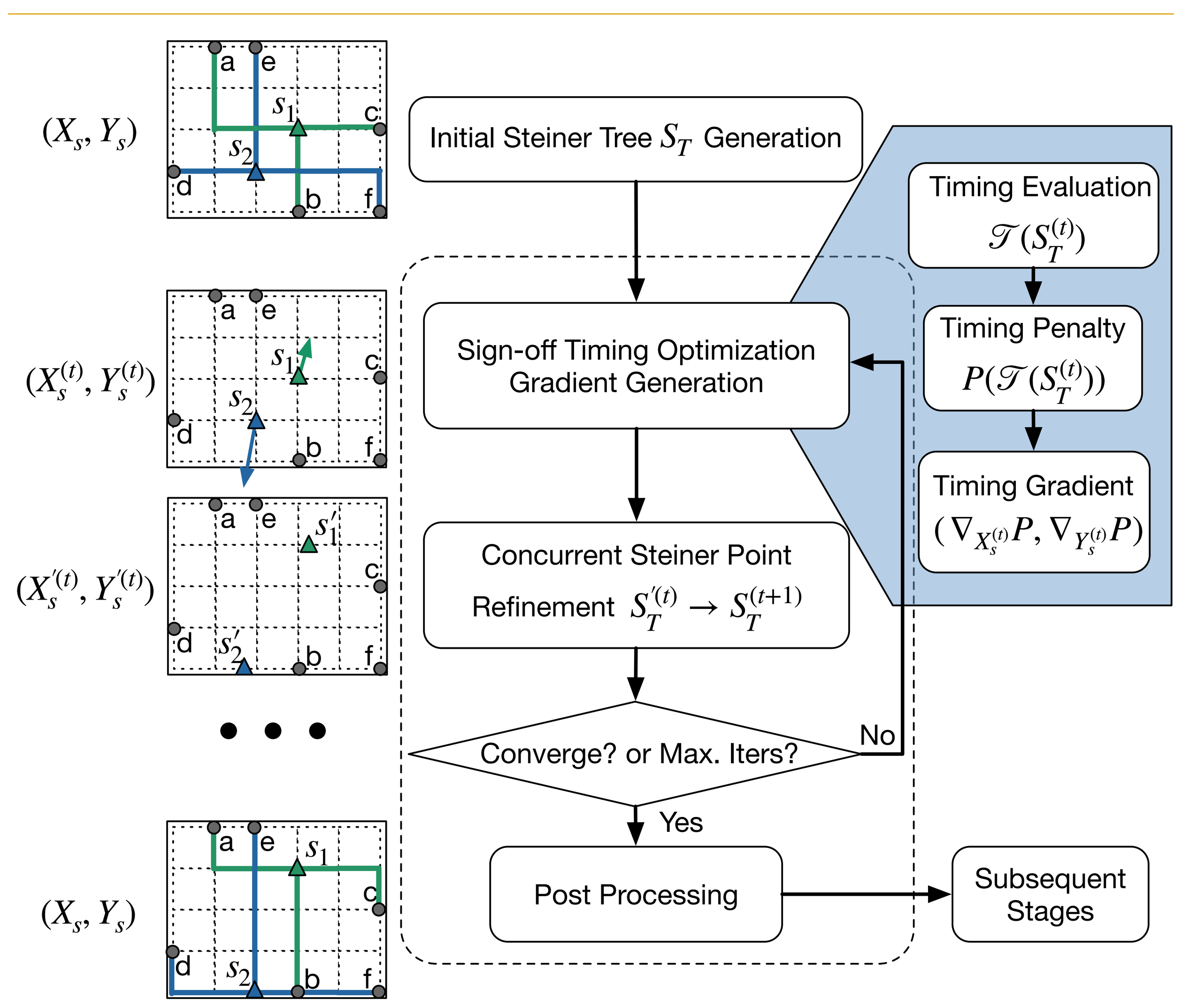


Highlights

- The sign-off timing performance could be significantly affected even by a random disturbance on Steiner point position.
- The impact of random moving is considerately unstable, and its average performance is slight.

Timing-driven Steiner point refinement Given an initial Steiner tree set $S_T = \{T^1, T^2, \cdots, T^n\}$, $T^i = (V_c^i, V_s^i, E^i)$, where V_c^i is the set of cell nodes, V_s^i is the set of Steiner nodes and E^i means the edges connecting V_c^i and V_s^i of the i^{th} Steiner tree, our task is to refine the position (X_s, Y_s) of $V_s = \{V_s^i, 1 \le i \le n\}$ in the pre-routing stage to obtain better **sign-off** timing performance.

Overall Flow - TSteiner



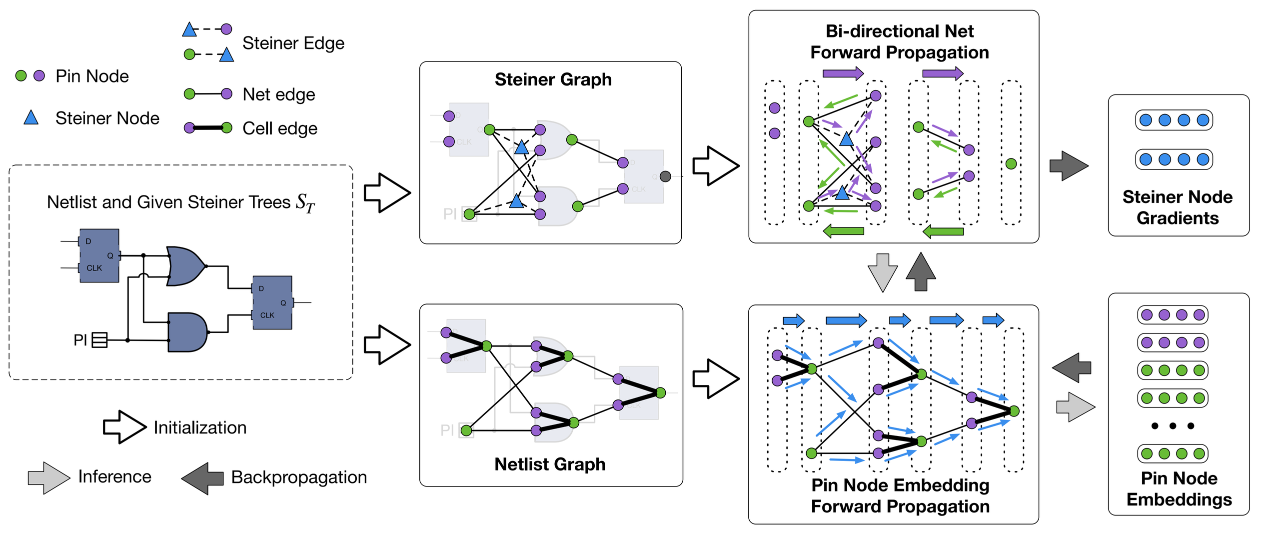
Highlights

- Sign-off timing optimization gradient $(\nabla_{X_s}P, \nabla_{Y_s}P)$ generation using timing evaluation model.
- Concurrent Steiner point refinement optimizes the Steiner points based on $(\nabla_{X_s}P, \nabla_{Y_s}P)$.

Evaluation Model

Sign-off Timing Evaluation Given a Steiner tree solution S_T, timing evaluation is to find an estimator $\mathcal{T}$ to evaluate the sign-off timing metrics $\mathcal{T}(S_T)$, i.e., arrival time at each pin.

Evaluation Model



Highlights

- Node-, Net-, and Graph- Heterogenous.
- Bi-directional Net Forward Propagation and Pin Node Embedding Forward Propagation.

Timing Penalty

Timing penalty can be calculated as,

$$P(\mathcal{T}(S_T)) = \lambda_w w(\mathcal{T}(S_T)) + \lambda_t t(\mathcal{T}(S_T)), \tag{1}$$

Replace the maximum operation with the Log-Sum-Exp function LSE,

$$LSE(x_1, x_2, \cdots, x_n) = \gamma \log\left(\sum_{i=1}^{n} \exp\frac{x_i}{\gamma}\right), \tag{2}$$

the smoothed penalty function can be expressed as,

$$P_\gamma(\mathcal{T}(S_T)) = \lambda_w w_\gamma(\mathcal{T}(S_T)) + \lambda_t t_\gamma(\mathcal{T}(S_T)). \tag{3}$$

Steiner Point Optimization

Stochastic Optimization Algorithm $SO(X_s^{(t)}, \nabla_{X_s^{(t)}}P)$

$$m_x^{(t)} = (1-\beta_1)\cdot\nabla_{X_s^{(t)}}P, \quad v_x^{(t)} = (1-\beta_2)\cdot(\nabla_{X_s^{(t)}}P \odot \nabla_{X_s^{(t)}}P),$$
$$X_s'^{(t)} = X_s^{(t)} - \theta \cdot \frac{m_x^{(t)}}{\sqrt{v_x^{(t)}} + \epsilon}, \tag{4}$$

Adaptive Stepsize Scheme *Adaptive_Theta*

1. Obtain the initial timing gradient $(\nabla_{X_s}P, \nabla_{Y_s}P)$ w.r.t. the given Steiner point positions (X_s, Y_s).
2. Apply a small move:
$$X_s' = X_s + \alpha\nabla_{X_s}P, \quad Y_s' = Y_s + \alpha\nabla_{Y_s}P, \tag{5}$$
where α is a hyper-parameter to control the scale of θ.
3. Obtain the updated timing gradient $(\nabla_{X_s'}P, \nabla_{Y_s'}P)$.

The adaptive stepsize is then calculated as:

$$\theta = \frac{\|(X_s, Y_s) - (X_s', Y_s')\|_2}{\|(\nabla_{X_s}P, \nabla_{Y_s}P) - (\nabla_{X_s'}P, \nabla_{Y_s'}P)\|_2}. \tag{6}$$

Concurrent Steiner Point Refinement

Input: S_T: initial Steiner trees; $\mathcal{T}$: pre-trained timing prediction model; N: maximum optimization iterations; μ: converge ratio.

```
init_wns ← w(T(S_T)); best_wns ← w(T(S_T));
init_tns ← t(T(S_T)); best_tns ← t(T(S_T));
θ ← Adaptive_Theta(S_T); t ← 0; S_T^(0) ← S_T; X_s^(0) ← X_s; Y_s^(0) ← Y_s;
Initialize the optimizer SO with θ;
repeat
    S'_T^(t) ← SO(S_T^(t), (∇_{X_s^(t)} P, ∇_{Y_s^(t)} P));
    wns ← w(T(S'_T^(t))); tns ← t(T(S'_T^(t)));
    if wns > best_wns or tns > best_tns then
        best_wns ← wns; best_tns ← tns; S_T^(t+1) ← S'_T^(t);
    else
        S_T^(t+1) ← S_T^(t);
    end if
    t ← t + 1;
    if t ≥ N then
        break;
    end if
until (init_wns − best_wns)/init_wns > μ or (init_tns − best_tns)/init_tns > μ

return S_T^(t) (Resulting Steiner Trees)
```

Results

Table 1. Sign-off Timing prediction performance on two tasks, where 'arrival-all' and 'arrival-ends' represent the arrival time prediction on all pins and only endpoints, respectively.

Benchmark	chacha	cic_decimator	APU	des	jpeg_encoder	spm	Avg. Train
arrival-all	0.9882	0.9980	0.9950	0.9989	0.9959	0.9991	0.9959
arrival-ends	0.9979	0.9990	0.9977	0.9976	0.9936	0.9987	0.9974

Benchmark	aes_cipher	picorv32a	usb_cdc_core	des3	Avg. Test
arrival-all	0.9468	0.9401	0.9163	0.9087	0.9280
arrival-ends	0.9459	0.9498	0.7015	0.9510	0.8871

Table 2. Comparison with the routing flow without integrating TSteiner.

Benchmark	CUGR [1] + TritonRoute [2]						TSteiner + CUGR [1] + TritonRoute [2]					
	WNS (ns)	TNS (ns)	# Vios	WL($\times 10^6$)	# Vias	# DRV	WNS (ns)	TNS (ns)	# Vios	WL($\times 10^6$)	# Vias	# DRV
aes_cipher	-11.246	-1516.9	512	984.971	109574	5	**-8.38**	**-1434.2**	**504**	984.527	109443	**3**
chacha	-48.538	-26259.1	1378	1,257.427	126600	2	**-46.68**	**-25375.7**	**1372**	1,258.011	126898	2
cic_decimator	-2.834	-169.981	72	16.466	5586	3	**-2.724**	**-161.436**	72	16.413	5593	3
picorv32a	-17.762	-441.607	67	727.216	109293	38	**-17.686**	**-434.443**	**56**	727.472	109311	**37**
usb_cdc_core	-5.914	-1365.2	347	49.351	12396	0	**-5.823**	**-1343.1**	**346**	49.117	12407	0
APU	-2.265	-33.713	25	101.179	23031	3	**-2.221**	**-33.598**	25	101.454	23101	3
des	-7.352	-405.427	341	682.828	115698	5	**-3.987**	**-227.331**	**285**	682.788	115599	5
jpeg_encoder	-74.342	-64909.2	**1967**	2,969.654	439126	1	**-70.629**	**-60789.1**	2007	2,973.304	439561	1
des3	-7.048	-1890	1512	2,680.848	372583	**48**	**-5.668**	**-1879.6**	**1509**	2,684.367	372768	49
spm	-0.817	-65.866	126	4.394	1553	2	**-0.782**	**-63.846**	126	4.399	1544	2
Average	1.000	1.000	1.000	1.0000	**1.0000**	1.0000	**0.888**	**0.929**	**0.967**	**0.9999**	1.0001	**0.9549**

Table 3. Runtime (s) breakdown

Benchmark	[1] + [2]			TSteiner + [1] + [2]			
	Total	[1]	[2]	Total	TSteiner	[1]	[2]
aes_cipher	539.847	16.781	523.066	528.512	22.222	17.830	488.460
chacha	480.123	19.795	460.328	504.902	60.331	21.642	422.929
cic_decimator	133.794	0.592	133.202	143.821	20.174	0.529	123.118
picorv32a	406.286	12.622	393.664	472.972	104.957	13.109	354.906
usb_cdc_core	54.660	1.247	53.413	115.784	59.515	1.156	55.113
APU	95.120	2.299	92.821	120.993	33.563	2.479	84.951
des	243.901	10.725	233.176	280.599	54.075	11.423	215.101
jpeg_encoder	893.485	61.433	832.052	1098.426	215.533	62.916	819.977
des3	558.014	37.863	520.151	941.474	414.908	40.841	485.725
spm	11.848	0.307	11.541	17.493	7.195	0.279	10.019
Ratio Avg.	1.000	1.000	1.000	1.320		1.017	0.934

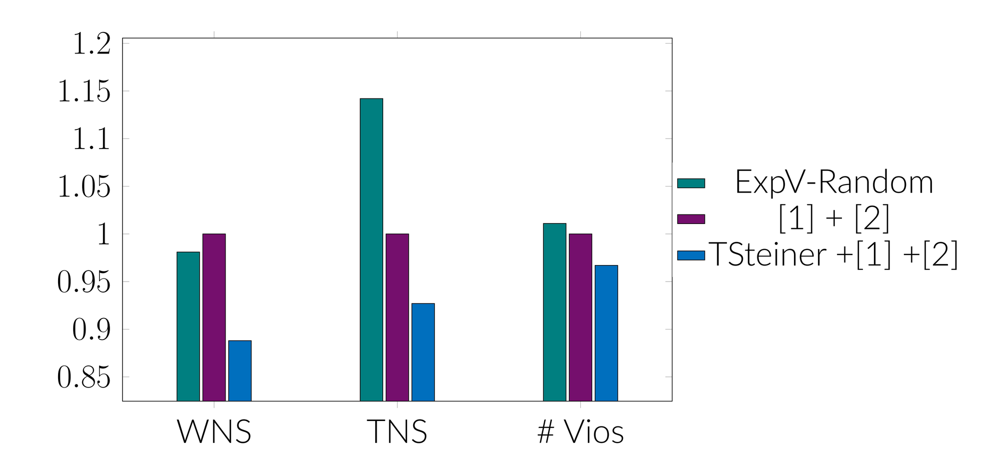


Figure 1. Sign-off timing metrics ratio comparison. 'ExpV-Random' represents the average expected value with 10-50 times random moves.

Highlights

- Our proposed timing evaluation model can accurately predict sign-off timing metrics.
- TSteiner brings **11.2%** and **7.1%** on average and up to **45.8%** and **43.9%** improvement for WNS and TNS, respectively.
- TSteiner brings only a little runtime overhead and comparable routing solution quality.

Conclusion & Future Directions

- This study has raised the importance of Steiner point refinement for timing closure and provides a novel solution for early-stage timing optimization.
- TSteiner can be extended to more physical stages since Steiner points exist not only in the pre-routing stage but also in routing solutions.

References

[1] J. Liu, C.-W. Pui, F. Wang, and E. F. Young, "CUGR: Detailed-routability-driven 3d global routing with probabilistic resource model," in *Proc. DAC*, pp. 1–6, 2020.

[2] A. B. Kahng, L. Wang, and B. Xu, "Tritonroute: The open-source detailed router," *IEEE TCAD*, vol. 40, no. 3, pp. 547–559, 2020.