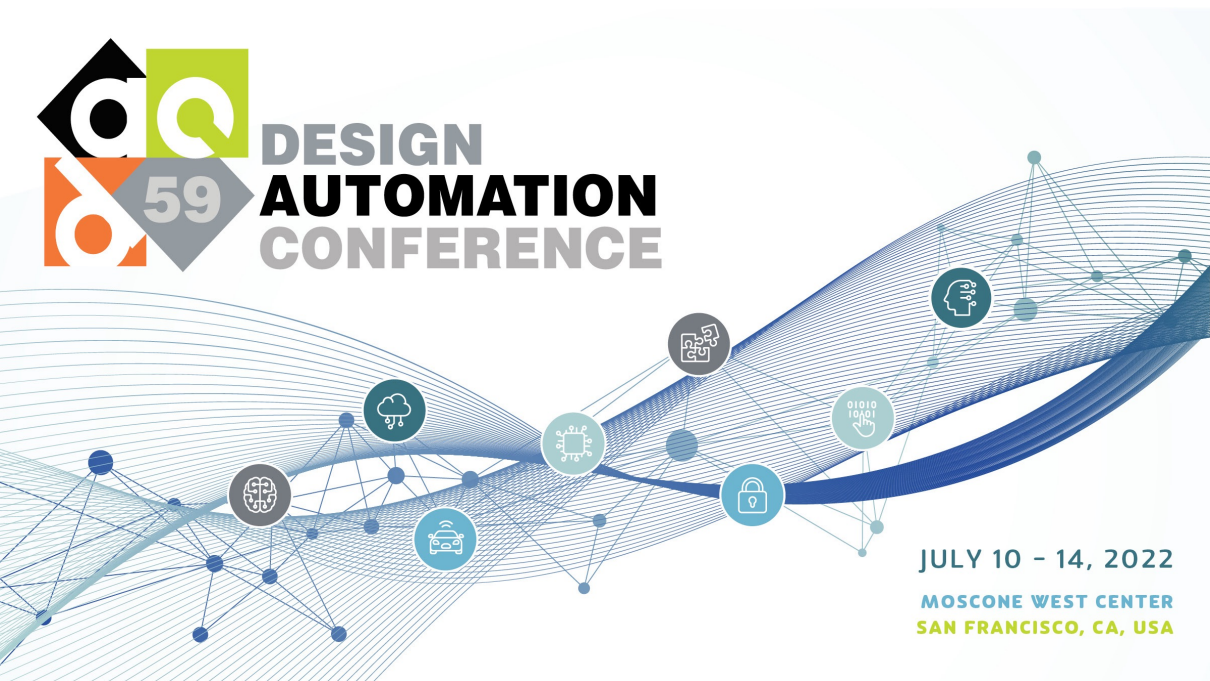

59
DESIGN
AUTOMATION
CONFERENCE
JULY 10 – 14, 2022
MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA

DESIGN AUTOMATION CONFERENCE 59

PPATuner: Pareto-driven Tool Parameter Auto-tuning in Physical Design via Gaussian Process Transfer Learning

Hao Geng[1,2], Qi Xu[3], Tsung-Yi Ho[2], Bei Yu[2]

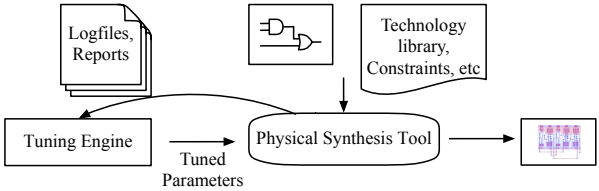
[1]ShanghaiTech [2]CUHK [3]USTC

{genghao}@shanghaitech.edu.cn

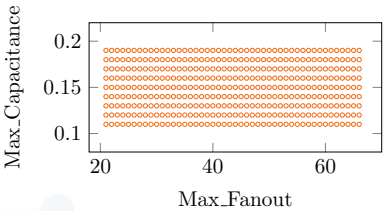
{byu}@cse.cuhk.edu.hk

July, 2022

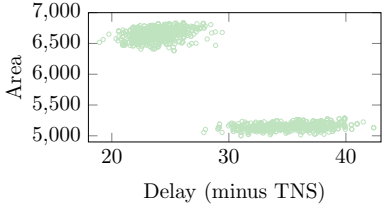
What is Physical Design (PD) Tool Parameter Tuning?



(a) PD tool parameter tuning



(b) The design space



(c) The QoR metric space

General Issues in Parameter Tuning

- A lot of values of design parameters need to be determined or tuned
- **Multiple** quality-of-result (QoR) metrics (e.g., area, power, and delay) to be optimized
- "**Black-box**" parameter-to-performance mappings: **no** explicit function expressions
- **Time-consuming** EDA tool evaluation, i.e., expensive data annotation
- Some existing tuning works:
  1. DAC'19 [1] based on neural network and tensor decomposition requires much dataset for start
  2. ASPDAC'20 [2] exploiting weight-sum trick to handle multiple QoR metrics.

Special Issues and Our Motivations (i)

To search good parameter configurations and reduce the cost on tool evaluation, we may consider ...

- developing more efficient searching framework (e.g., heuristics, Bayesian Optimization, and etc.)
- exploiting the historical dataset since similar parameter configurations tried quite a few times in navigation of prior tasks

Special Issues and Our Motivations (ii)

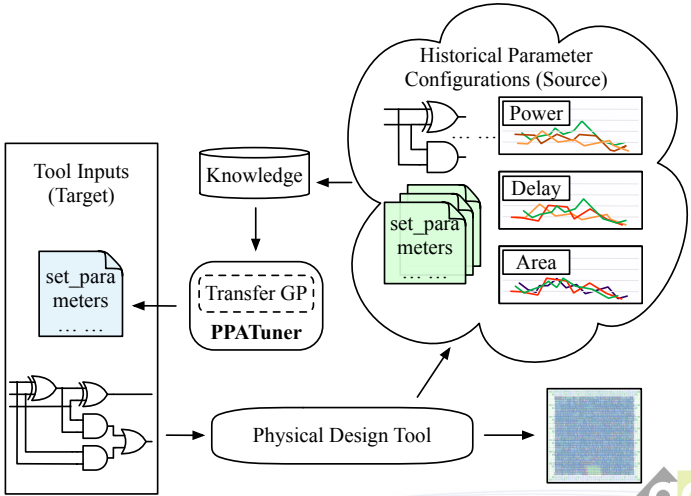
However, issues merge ...

- how to design the searching framework?
- how to use the historical dataset?
  1. given the same design, how to exploit the current framework when considering different parameter configurations?
  2. how to transfer the current framework from small designs (tool evaluation is relatively cheap) to big designs?

Our solution: Overview

- A Pareto-driven iterative auto-tuning framework
  1. Able to handle multi-objective optimization problem
  2. Not requiring too many samples for tool evaluation when keeping the good quality of QoR metrics of predicted points
  3. Small runtime overhead
- Gaussian process with transfer learning utilizing the prior data

Our solution: Visualization



Our solution: Transfer Gaussian Process (i)

- Main idea: learn **a transfer kernel** to model the correlation of the outputs when the inputs come from different tasks, namely, a measure of similarity between tasks.
- Assume we have N tool parameter configurations for the source task and M instances for the target, then $f^{\mathcal{S}}$ is an N-dimensional vector and $f^{\mathcal{T}}$ an M-dimensional vector.
- Define a Gaussian process over $\mathbf{f} = \left(\mathbf{f}^{(\mathcal{S})}, \mathbf{f}^{(\mathcal{T})}\right)$:

$$p(\mathbf{f} \mid \mathbf{X}, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{f} \mid \mathbf{0}, \mathbf{K}) \tag{1}$$

where $\boldsymbol{X}$ refers to the tool parameter configurations and $\boldsymbol{\theta}$ represents the whole hyper-parameters with the **kernel matrix K** for transfer learning $\boldsymbol{K}_{nm} \sim k\left(\boldsymbol{x}_n, \boldsymbol{x}_m\right)\left(2e^{-\eta\left(\boldsymbol{x}_n, \boldsymbol{x}_m\right)\phi} - 1\right)$, where $\eta\left(\boldsymbol{x}_n, \boldsymbol{x}_m\right) = 0$ if $\boldsymbol{x}_n$ and $\boldsymbol{x}_m$ are from the same task domain, otherwise $\eta\left(\boldsymbol{x}_n, \boldsymbol{x}_m\right) = 1$. The parameter ϕ represents the dissimilarity between $\mathcal{S}$ and $\mathcal{T}$.

Our solution: Transfer Gaussian Process (ii)

- Given a test parameter configuration $\boldsymbol{x}$ in the target task, the mean and variance of the predictive distribution of it are

$$
\begin{aligned}
u(\boldsymbol{x}) &= k(\boldsymbol{x}, \boldsymbol{X})^{\top} \left(\widetilde{\boldsymbol{K}} + \boldsymbol{\Lambda}\right)^{-1} \boldsymbol{y}, \\
\sigma^2(\boldsymbol{x}) &= c - k(\boldsymbol{x}, \boldsymbol{X})^{\top} \left(\widetilde{\boldsymbol{K}} + \boldsymbol{\Lambda}\right)^{-1} k(\boldsymbol{x}, \boldsymbol{X}),
\end{aligned}
\tag{2}
$$

where $\boldsymbol{\Lambda} = \begin{pmatrix} \beta_s^{-1}\boldsymbol{I}_N & \boldsymbol{0} \\ \boldsymbol{0} & \beta_t^{-1}\boldsymbol{I}_M \end{pmatrix}$ and $c = k(\boldsymbol{x}, \boldsymbol{x}) + \beta_t^{-1}$ and $k(\boldsymbol{x}, \boldsymbol{X})$ can be computed by the transfer kernel. β_s and β_t are two hyper-parameters which can be learned by maximizing the marginal likelihood of data of the target task during training.

Our solution: The Proposed PPATuner

Algorithm 1 The PPATuner

Input: the historical parameter configuration dataset $\mathcal{D}^{\mathcal{S}}$, the target parameter configuration dataset $\mathcal{D}^{\mathcal{T}}$ for initialization, the number of maximum iterations T_{max}.
Output: the predicted Pareto-optimal parameter configuration set.

 # Initialization:
Initializing the transfer GP models with $\mathcal{D}^{\mathcal{S}}$ and $\mathcal{D}^{\mathcal{T}}$;
while existing not decided configurations or $t < T_{max}$ **do**
    # Model Calibration:
    The transfer GP models further calibration with $\boldsymbol{x}_t^s$ and associated golden QoR metric values;
    Uncertainty regions construction;
    # Decision-making:
    Dropping δ-dominated configurations;
    Pareto-optimal configurations deciding;
    # Selection:
    Choosing and sending the configurations $\boldsymbol{x}_t^s$ for the PD tool evaluation;
    $t \leftarrow t + 1$;
end while

The Benchmarks and Experimental Settings

- Implementation based on `Python` and Cadence Innovus Implementation System as the PD tool.
- The experimental platform is a Xeon Silver with the 4114 CPU processor
- Scenario One: Same Design:
  - Source1: an industrial benchmark generated by a small MAC design (20k cells) under 7nm node (5000 tool parameter configurations with 12 parameters)
  - Target1: an industrial benchmark generated by the same MAC design under 7nm node (5000 tool parameter configurations with different 12 parameters)
- Scenario Two: Similar Designs:
  - Source2: an industrial benchmark generated by a small MAC design (20k cells) under 7nm node (1440 tool parameter configurations with 9 parameters)
  - Target2: an industrial benchmark generated by a larger MAC design (67k cells) under 7nm node (727 tool parameter configurations with 9 parameters)

The Benchmark Statistics

Table: The statistics of parameters of the PD tool on benchmarks. “-" in the table means the parameter is not considered in this benchmark.

Parameters	Source1		Target1		Source2		Target2	
	Min	Max	Min	Max	Min	Max	Min	Max
freq	950	1050	1000	1300	-	-	-	-
place_rcfactor	-	-	-	-	1.00	1.30	1.00	1.30
place_uncertainty	50	200	20	100	-	-	-	-
flowEffort	standard	extreme	standard	extreme	standard	extreme	standard	extreme
timing_effort	-	-	-	-	medium	high	medium	high
clock_power_driven	-	-	-	-	FALSE	TRUE	FALSE	TRUE
uniform_density	FALSE	TRUE	FALSE	TRUE	-	-	-	-
cong_effort	AUTO	HIGH	AUTO	HIGH	-	-	-	-
max_density	0.65	0.90	0.65	0.90	-	-	-	-
max_Length	160	310	160	300	250	350	250	350
max_Density	0.65	0.90	0.65	0.90	0.50	1.00	0.50	1. 00
max_transition	0.19	0.34	0.10	0.35	-	-	-	-
max_capacitance	0.08	0.13	0.08	0.20	0.07	0.12	0.05	0.15
max_fanout	25	50	25	50	25	40	25	39
max_AllowedDelay	0.00	0.25	0.00	0.25	0.06	0.12	0.00	0.12

The Experimental Results (i)

Table: The whole performance comparison on Target1 benchmark.

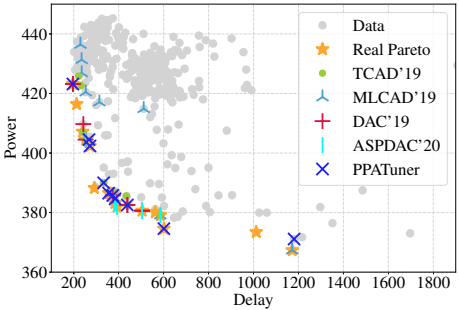
Multi-objective	TCAD'19 [3]			MLCAD'19 [4]			DAC'19 [1]			ASPDAC'20 [2]			PPATuner		
	HV	ADRS	Runs	HV	ADRS	Runs	HV	ADRS	Runs	HV	ADRS	Runs	HV	ADRS	Runs
Area-Delay	0.142	0.068	509	0.122	0.061	400	0.129	0.075	627	0.133	0.063	400	0.050	0.041	251
Power-Delay	0.237	0.235	512	0.199	0.256	400	0.243	0.266	596	0.190	0.178	400	0.095	0.099	259
Area-Power-Delay	0.185	0.064	503	0.160	0.058	400	0.214	0.100	577	0.195	0.086	400	0.096	0.075	247
Average	0.188	0.122	508	0.160	0.125	400	0.195	0.147	600	0.173	0.109	400	**0.080**	**0.072**	**252.333**
Ratio	2.350	1.694	2.013	2.000	1.736	1.585	2.438	2.042	2.378	2.163	1.514	1.585	**1.000**	**1.000**	**1.000**

The Experimental Results (ii)

Table: The whole performance comparison on Target2 benchmark.

Multi-objective	TCAD'19 [3]			MLCAD'19 [4]			DAC'19 [1]			ASPDAC'20 [2]			PPATuner		
	HV	ADRS	Runs	HV	ADRS	Runs	HV	ADRS	Runs	HV	ADRS	Runs	HV	ADRS	Runs
Area-Delay	0.103	0.099	95	0.140	0.104	70	0.133	0.092	132	0.114	0.116	70	0.053	0.049	65
Power-Delay	0.113	0.092	93	0.121	0.094	70	0.127	0.077	125	0.122	0.119	70	0.039	0.050	63
Area-Power-Delay	0.107	0.084	88	0.099	0.074	70	0.105	0.103	137	0.140	0.086	70	0.059	0.042	58
Average	0.108	0.092	92	0.120	0.091	70	0.122	0.091	131.333	0.125	0.107	70	**0.050**	**0.047**	**62**
Ratio	2.160	1.957	1.484	2.400	1.936	1.129	2.440	1.936	2.118	2.500	2.277	1.129	**1.000**	**1.000**	**1.000**

The Experimental Results (iii)



The visualization of Pareto frontiers in power vs. delay space on Target2 benchmark. The units for power and delay are mW and ns, respectively.

References I

[1] J. Kwon, M. M. Ziegler, and L. P. Carloni, "A learning-based recommender system for autotuning design fiows of industrial high-performance processors", in *Proc. DAC*, 2019, pp. 1–6.

[2] Z. Xie, G.-Q. Fang, Y.-H. Huang, H. Ren, Y. Zhang, B. Khailany, S.-Y. Fang, J. Hu, Y. Chen, and E. C. Barboza, "FIST: A feature-importance sampling and tree-based method for automatic design flow parameter tuning", in *Proc. ASPDAC*, 2020, pp. 19–25.

[3] Y. Ma, S. Roy, J. Miao, J. Chen, and B. Yu, "Cross-layer optimization for high speed adders: A Pareto driven machine learning approach", *IEEE TCAD*, vol. 38, no. 12, pp. 2298–2311, 2019.

[4] Y. Ma, Z. Yu, and B. Yu, "CAD tool design space exploration via Bayesian optimization", in *Proc. MLCAD*, 2019, pp. 1–6.

THANK YOU!