DREAMPlace 4.0: Timing-driven Global Placement with Momentum-based Net Weighting

Peiyu Liao[1,2], Siting Liu[1,2], Zhitang Chen[3], Wenlong Lv[3], Yibo Lin[1*], Bei Yu[2*]
[1]Peking University [2]Chinese University of Hong Kong [3]Huawei Noah's Ark Lab

***Abstract*—Timing optimization is critical to integrated circuit (IC) design closure. Existing global placement algorithms mostly focus on wirelength optimization without considering timing. In this paper, we propose a timing-driven global placement algorithm leveraging a momentum-based net weighting strategy. Besides, we improve the preconditioner to incorporate our net weighting scheme. Experimental results on ICCAD 2015 contest benchmarks demonstrate that our algorithm can significantly improve total negative slack (TNS) and meanwhile be beneficial to worse negative slack (WNS).**

I. Introduction

Circuit placement is a challenge of finding good locations for individual circuit components [1]. Since placement is only an interior stage of chip design, it is important to define the evaluation and target of placement. Most of the previous placement algorithms focus on minimizing total wirelength. However, in the placement stage, a wirelength model is only for an approximation so it is almost impossible to make it accurate. Besides, a general focus on the total wirelength will simply ignore those timing-critical paths. In contrast, timing-driven placement is designed specifically targeting wires on these timing-critical paths.

Timing optimization can be performed in both global and detailed placement stages. Timing-driven global placement aims at achieving roughly good *total negative slack* (TNS) and *worst negative slack* (WNS), while timing-driven detailed placement usually focuses on WNS optimization by performing local perturbation to the current placement solution. Timing optimization in placement can be categorized into *net-based* approaches and *path-based* approaches.

Net-based approaches care about nets in the design. They try to optimize timing by translating feedbacks of timing analysis into net weights or constraints. Static net weighting, including *slack-based* [2]–[5] and sensitivity-based [6]–[8] approaches, compute net weights only once before timing-driven placement. However, during global placement, cell locations are unreliable for us to perform effective timing analysis at earlier stages. On the other hand, they are also very likely to change significantly and thus make static net weighting ineffective. Dynamic net weighting [2], [9]–[11] updates net weights gradually, with placement solution at different iterations considered. Net constraint-based approaches [12]–[16] limit the maximum length of specific nets. The formulation of net constraints varies from particular placers [1].

Path-based approaches [17]–[20] directly work on paths instead. They aim at explicitly reducing the delay of selected paths by moving cells. Path-based approaches usually formulate the optimization problem as mathematical programming and can achieve better qualities compared to net-based approaches. However, a large design may introduce a large number of paths, and therefore the run-time issue should be considered seriously in the analysis of path-based approaches.

Both net-based and path-based approaches have pros and cons regarding different metrics. The tradeoff is unavoidable for designers to optimize timing. Unlike detailed placement, the cell locations have much more freedom during global placement. Therefore, we prefer relatively generic approaches, can incorporate updates of cell locations at different iterations, and do not have intolerable runtime issues.

In this paper, we propose a timing-driven global placement engine with momentum-based net weighting. We choose the net-based approach for its scalability to global perturbation of cells in global placement. The major contributions are summarized as follows.

- **Momentum-based net weighting**. The weighting scheme plays an important role in our timing-driven global placement algorithm. At each timing iteration, we expect to assign weights to different nets by incorporating the current slacks within the existing criticality information. The net weights will be updated gradually by considering the new weights, computed according to slacks, to be a momentum term, which is analogous to the *momentum method* that is widely used in backpropagation learning [21].
- **Preconditioning technique for net weighting**. The preconditioner proposed by the original ePlace [22] algorithm does not consider different net weights. Considering that we may assign very different net weights to different nets to optimize timing, the numerical stability may get negatively affected, especially for those cells incident to critical paths. We enhance our preconditioner to adapt different net weights when optimizing cell locations.
- Experimental results on the ICCAD2015 contest benchmark suites [23] show that on average we can achieve 46.83% improvements on TNS, and 30.27% improvements on WNS, compared to the state-of-the-art placer [24] after global placement and legalization.

The rest of the paper is organized as follows. Section II provides some preliminaries including brief foundations of nonlinear placement, static timing analysis, and timing optimization. Section III presents the overall flow of our timing-driven global placement algorithms and the detailed explanations. Section IV demonstrates the experimental results and some related analysis, followed by Section V summarizing the whole paper.

*Corresponding authors: yibolin@pku.edu.cn, byu@cse.cuhk.edu.hk

II. Preliminary

A. Nonlinear Global Placement

At the global placement stage, a given circuit is considered to be a graph where vertices model gates. Placers are expected to place millions of instances to appropriate locations such that the total wirelength can be minimized. The netlist $\mathcal{N} = (E, V)$ consists of a net set E and a node (cell) set V. Suppose that we have n nodes in the design (*i.e.* $|V| = n$), the global placement seeks locations $(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}^n \times \mathbb{R}^n$ that minimize the total half-perimeter wirelength $W(\boldsymbol{x}, \boldsymbol{y})$. If we assign a net weight w_e for each net $e \in E$, the optimization formulation can be modified to minimizing the weighted sum

$$\min_{\boldsymbol{x},\boldsymbol{y}} \sum_{e \in E} w_e W(e; \boldsymbol{x}, \boldsymbol{y}). \tag{1}$$

There are many modern techniques to smoothly approximate the half-perimeter wirelength model $W(e; \boldsymbol{x}, \boldsymbol{y})$. Nonlinear placement adopts a *nonlinear* differentiable approximation of $W(e; \boldsymbol{x}, \boldsymbol{y})$. A widely-used approximation is the weighted-average (WA) model [25], [26],

$$\tilde{W}_x(e, \gamma; \boldsymbol{x}, \boldsymbol{y}) = \frac{\sum_{i \in e} x_i \mathrm{e}^{\frac{x_i}{\gamma}}}{\sum_{i \in e} \mathrm{e}^{\frac{x_i}{\gamma}}} - \frac{\sum_{i \in e} x_i \mathrm{e}^{-\frac{x_i}{\gamma}}}{\sum_{i \in e} \mathrm{e}^{-\frac{x_i}{\gamma}}}, \tag{2}$$

$$\tilde{W}(e, \gamma; \boldsymbol{x}, \boldsymbol{y}) = \tilde{W}_x(e, \gamma; \boldsymbol{x}, \boldsymbol{y}) + \tilde{W}_y(e, \gamma; \boldsymbol{x}, \boldsymbol{y}),$$

where $\tilde{W}_x(e, \gamma; \boldsymbol{x}, \boldsymbol{y})$ and $\tilde{W}_y(e, \gamma; \boldsymbol{x}, \boldsymbol{y})$ are the net wirelength along horizontal and vertical direction, respectively. γ is a hyperparameter to control the precision of this approximation. A typical non-linear placement problem can be formulated as

$$\min_{\boldsymbol{x},\boldsymbol{y}} \sum_{e \in E} w_e W(e; \boldsymbol{x}, \boldsymbol{y}) + \lambda D(\boldsymbol{x}, \boldsymbol{y}), \tag{3}$$

where E is the net set, $W(e; \cdot, \cdot)$ is the wirelength function that calculates the total wirelength of a specific net $e \in E$, function $D(\cdot, \cdot)$ indicates the total density penalty and λ is the corresponding density weight. This is a typical *unconstrained* optimizatin problem with arguments $\boldsymbol{x}, \boldsymbol{y}$ being the cell locations.

The objective function (3) is required to be everywhere differentiable so that we can use gradient-based methods to optimize the variables. Additionally, the term $W(e; \boldsymbol{x}, \boldsymbol{y})$ is a nonlinear approximation of the net wirelength.

B. Static Timing Analysis

The timing-driven placement has to be guided by timing analysis. Static timing analysis (STA) evaluates the setup/hold timing performance of a circuit under best-case and worst-case scenarios based on its delay-annotated timing graph [27], [28]. It performs forward and backward propagation to compute the arrival time and the required arrival time for each node in the graph, respectively [29].

More specifically, we model the given circuit as a *directed acyclic graph* (DAG). Each node in the DAG corresponds to a pin in the circuit, and each edge in this graph represents a directed pin-to-pin connection. A complete STA process evaluates the delays of nets and cells, and then computes the arrival time and required arrival time of pins through propagation. For a specific pin p, assume that we have its arrival time $t_{\text{at}}(p)$ and required arrival time $t_{\text{rat}}(p)$, then the *slack* of p is defined as the difference of its required arrival time minus arrival time,

$$s(p) = t_{\text{rat}}(p) - t_{\text{at}}(p). \tag{4}$$

Slack is an important metric to evaluate the timing quality of a placement solution. The *worst negative slack* (WNS) is the most commonly used timing metric, defined as the worst one among all negative slacks of timing endpoints,

$$s_{\text{wns}} = \min_{t \in P_{\text{end}}} s(t), \tag{5}$$

where P_{end} indicates the set of all timing endpoints, and s_{wns} is the worst negative slack. We assume that there exists at one $t \in P_{\text{end}}$ such that $s(t) < 0$, otherwise the timing constraints are perfectly satisfied. Another well-known timing closure objective is the *total negative slack* (TNS), defined as the sum of all negative slacks of timing endpoints,

$$s_{\text{tns}} = \sum_{t \in P_{\text{end}}, s(t) < 0} s(t), \tag{6}$$

where s_{tns} stands for the total negative slack [16].

C. Timing Optimization

Timing-driven placement pays more attention to timing optimization. Rather than the total wirelength of the circuit design, we prefer objectives that are more suitable to reflect timing metrics. TNS and WNS are both well-adopted timing metrics, however, they may emphasize different aspects. Intuitively, WNS may only provide timing information of a single critical path, while TNS gives the overall timing information of multiple or even a large number of critical paths. Empirically, TNS should be more important to guide the timing optimization during global placement, as it can integrate information of all critical paths. On the contrary, WNS should be a more important metric when optimizing timing precisely in detailed placement.

The complete formulation of timing-driven global placement can be summarized as follows.

$$\begin{aligned} \max \quad & s(\boldsymbol{x}, \boldsymbol{y}) \\ \text{s.t.} \quad & \rho_b(\boldsymbol{x}, \boldsymbol{y}) \leq \rho_t, \forall b \in B, \end{aligned} \tag{7}$$

where B is the set of $m \times m$ planar grids (bins) for a positive integer m, $\rho_b(\boldsymbol{x}, \boldsymbol{y})$ denote the density of a bin $b \in B$, ρ_t represents the target placement density of each bin, and the objective function $s(\boldsymbol{x}, \boldsymbol{y})$ stands for a *negative* slack function. Typically, the objective function $s(\boldsymbol{x}, \boldsymbol{y})$ can be TNS $s_{\text{tns}}(\boldsymbol{x}, \boldsymbol{y})$ or WNS $s_{\text{wns}}(\boldsymbol{x}, \boldsymbol{y})$. Equation (7) shares the same cell constraints but uses a completely different objective function from that of wirelength-driven analytical placement. Unlike wirelength functions that usually have closed-form representations with respect to cell locations directly, slack functions cannot be represented explicitly. That is the reason why we decide to optimize timing indirectly by implementing net weighting schemes in the wirelength optimization.

III. Algorithms

The overall flow of our placement framework with timing analysis is illustrated in Fig. 1. Compared to modern gradient-based analytical placers, we must determine whether to perform timing analysis at each gradient-based iteration.

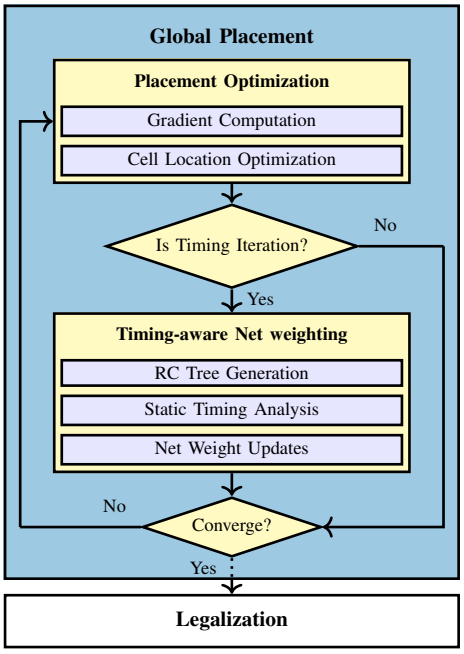
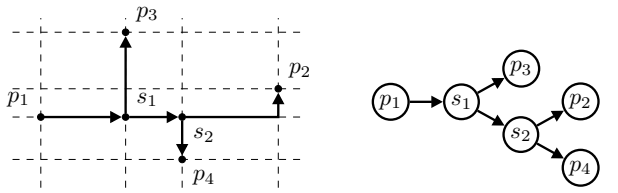


Fig. 1 Our overall flow with timing optimization.



(a) A Steiner tree example (b) The corresponding RC tree

Fig. 2 A 4-pin net example of Steiner tree and the corresponding RC tree constructed for net delay calculation.

A. RC Tree Construction

We must construct RC trees for nets manually at every timing iteration, as the cell locations are going to be changed in every backward step. Given a *possibly illegal* placement solution, we are provided with all pin locations for each net.

For a specific net, we start with the pin locations it contains. A FLUTE [30] call will be performed to construct the rectilinear Steiner minimal tree of this net. This Steiner tree generally reflects how the timing propagation will be performed internally inside the timer we use. We take a simple 4-pin net as an example to illustrate how to construct the RC tree for a certain net in Fig. 2(a) and Fig. 2(b). The abstract RC tree hierarchy is shown in Fig. 2(b). To construct RC information from interconnects, we require the resistance value per unit length r' and the capacitance value per unit length c', which should be pre-determined for the given design.

B. Delay Calculation

We can enrich details on Fig. 2(b) by adding some abstract resistors and capacitors for edges in the RC tree, illustrated in Fig. 3. Here the Elmore delay model [31] is used to approximate actual delays. More specifically, we use $\Pi-$model to break wires into RC sections. After we fill the RC information into the RC tree initialized by the timer, we then naturally proceed to the static timing analysis.

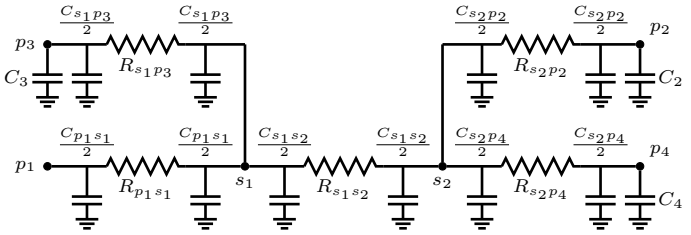


Fig. 3 The Elmore delay model for the above 4-pin net example.

C. Momentum-based Net Weighting

Net weights are assigned to all nets in the design so that some prior knowledge of how much contribution to the objective function these nets will make can be fed into the placer during global placement. The optimizer will implicitly get a stronger will to place cells containing pins included in nets with higher weights closer. Without any doubt, critical nets should be reasonably assigned higher weights to remind the placer to place cells related to them closer.

Net Criticality. Our placement database considers *criticality* value as a guide to update net weights. Let c_e and s_{wns} denote the criticality value of a specific net e and the worst negative slack, respectively. We can define the *momentum* of criticality value of a net e as

$$c_{\text{mom},e} = \begin{cases} 0, & \text{if } s_{\text{wns}} \geq 0; \\ \max\left\{0, \dfrac{s_e}{s_{\text{wns}}}\right\}, & \text{otherwise,} \end{cases} \tag{8}$$

where s_e is the net slack of e. If the worst negative slack s_{wns} is non-negative, everybody should be satisfied and we will do nothing to the net weights. Otherwise, $s_{\text{wns}} < 0$ is negative, and the criticality value is defined as the maximum value between 0 and the slack ratio s_e/s_{wns}.

If a net e has a non-negative slack $s_e \geq 0$, its criticality momentum will be set to $c_{\text{mom},e} = 0$, otherwise, it will be the ratio $\frac{s_e}{s_{\text{wns}}} = \frac{|s_e|}{|s_{\text{wns}}|}$. For net e, The higher slack value $|s_e|$ we obtain, the higher criticality momentum $c_{\text{mom},e}$ it will have.

Intuitively, the criticality indicates the probability of net e to be critical. Since we may obtain different timing-critical paths reported by the timer at each iteration, the criticality values should also be updated iteratively. A critical net may be related to multiple critical paths, and different nets may have different negative slacks. Hence, we are supposed to handle different critical nets in different ways. Nets with more negative slacks are considered to be more sensitive to timing metrics, and we should assign higher weights to them accordingly.

Define the criticality value of a net e at the m-th iteration as $c_e^{(m)}$. From Equation (8), we know that $s_e^{(m)}$ and $s_{\text{wns}}^{(m)}$, which represent the net slack and the WNS at the m-th iteration respectively, can be re-calculated at each timing iteration. Then we obtain a criticality value $c_e^{(m)}$ of net e at the m-th iteration, which corresponds to the criticality updates.

Net weighting Scheme. We introduce a momentum-based net weighting scheme. For a specific net e, let $\tilde{w}_e^{(m)} = \ln w_e^{(m)}$ and

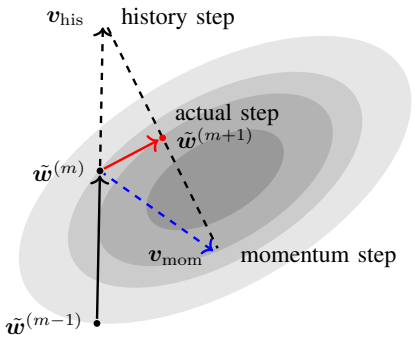


Fig. 4 A simple example illustrating how the momentum step vectors will affect the actual gradients.

$\Delta\tilde{w}_e^{(m)}$ be the logarithmic net weight of w_e and its increment at the m-th timing iteration, respectively.

$$\tilde{w}_e^{(m+1)} = \tilde{w}_e^{(m)} + \Delta\tilde{w}_e^{(m)}, \quad m \in \mathbb{N}. \tag{9}$$

The increment value $\Delta\tilde{w}_e^{(m)}$ is treated as the gradient determined by the timing metrics. Considering that we will obtain a new criticality *momentum* at each timing iteration. We expect the net weight $w_e^{(m)}$ to be emphsized by its criticality $c_e^{(m)}$. For integer $m \in \mathbb{N}$, the increment relationship can be modeled by

$$\begin{aligned} \Delta\tilde{w}_e^{(m)} &= \tilde{c}_e^{(m)}, \\ \Delta\tilde{w}_e^{(m+1)} &= \alpha\Delta\tilde{w}_e^{(m)} + (1-\alpha)\tilde{c}_{\text{mom},e}^{(m)}, \end{aligned} \tag{10}$$

where $\tilde{c}_e^{(m)} = \ln(1 + c_e^{(m)})$, $\tilde{c}_{\text{mom},e}^{(m)} = \ln(1 + c_{\text{mom},e}^{(m)})$ are the transformed criticality values and increments. The decay coefficient $\alpha \in [0, 1]$ is a hyperparameter. The term $\Delta\tilde{w}_e^{(m)}$ can be considered as the velocity, from Equation (9).

The scheme in Equations (9) and (10) is inspired by the momentum-based gradient descent algorithm on backpropagation during neural network training. In backpropagation, the momentum term should be the negative gradient of the objective, so that the actual gradient increment value can be guided while remembering the history update at each iteration. Here we apply the momentum step to the update of criticality value. If a net has a positive criticality value instead of zero, its weight should be increased according to the magnitude of criticality. Besides, if a net is reported to be critical at most timing iterations, it may have a large net weight. The weight differences are acceptable as long as no value overflow is reported.

We adopt the annotations in matrix calculus, then the scheme illustrated in Equations (9) and (10) can be reformulated as

$$\Delta\tilde{\boldsymbol{w}}^{(m+1)} = \alpha\Delta\tilde{\boldsymbol{w}}^{(m)} + (1-\alpha)\tilde{\boldsymbol{c}}_{\text{mom}}^{(m)}, \tag{11}$$

where $\tilde{\boldsymbol{w}}^{(m)}$, $\Delta\tilde{\boldsymbol{w}}^{(m)}$, and $\tilde{\boldsymbol{c}}_{\text{mom}}^{(m)}$ indicate the logarithmic net weights, their increments, and the transformed momentum vector calculated by Equation (8), respectively, at the m-th timing iteration. All these vectors have the same size that is exactly the total number of nets in the design. More specifically, the i-th entry of each of these vectors indicates an attribute of the net with index i. A simple example illustrating how momentum-based net weighting works is shown in Fig. 4.

If the momentum increment $\tilde{c}_{\text{mom},e}^{(m)}$ has a very small magnitude in the late period of global placement, the vector $\Delta\tilde{\boldsymbol{w}}^{(m)}$ will approximately decay by the factor α with the increment of iteration m, and correspondingly the net weight $\boldsymbol{w}^{(m)}$ will gradually stabilize. Therefore, we will keep emphasizing those nets that remain critical during placement.

Unlike [9] or other similar dynamic net weighting schemes, we work on all nets instead of those only on critical paths. Every net will be assigned a non-trivial criticality value related to its slack at a timing iteration, and then proceeds to the net weighting step.

D. Preconditioning

In numerical optimization, preconditioning is a very important step to reduce the condition number of an optimization problem. For a general unconstrained problem $\min_{\boldsymbol{x}} f(\boldsymbol{x})$, conventional preconditioning approaches aim at solving the inverse matrix of the Hessian $\boldsymbol{H}_f^{-1}$. Considering that the industrial designs may contain millions of instances, and such computation will have a huge overhead, the real implementation will become extremely unbearable.

The ePlace [22] preconditioner only considers diagonal entries of the Hessian matrix. The objective function f is set to Equation (3) by default. Without loss of generality, we only consider the horizontal direction here. Vector $\boldsymbol{x} \in \mathbb{R}^n$ represents the horizontal cell locations. The i-th diagonal entry of the Hessian matrix $\boldsymbol{H}_f^{-1}$ will be

$$\frac{\partial^2 f}{\partial x_i^2} = \sum_{e\in E} w_e \frac{\partial^2 W(e;\boldsymbol{x},\boldsymbol{y})}{\partial x_i^2} + \lambda \frac{\partial^2 D(\boldsymbol{x},\boldsymbol{y})}{\partial x_i^2}. \tag{12}$$

In the ePlace [22] algorithm, for any net $e \in E$, the term $\frac{\partial^2 W(e;\boldsymbol{x},\boldsymbol{y})}{\partial x_i^2}$ is simply binary. We also adopt this approximation. More specifically, only if the i-th node is incident to net e will the term be set to 1. This very rough evaluation will approximate the first term in Equation (12) as

$$\sum_{e\in E} w_e \frac{\partial^2 W(e;\boldsymbol{x},\boldsymbol{y})}{\partial x_i^2} \approx \sum_{e\in E_i} w_e, \tag{13}$$

where E_i is the net subset incident to the i-th node. Our net weighting scheme will not affect the density term, therefore we adopt the same approximation as [22] for preconditioning.

$$\frac{\partial^2 D(\boldsymbol{x},\boldsymbol{y})}{\partial x_i^2} = q_i \frac{\partial^2 \phi_i(\boldsymbol{x},\boldsymbol{y})}{\partial x_i^2} \approx q_i, \tag{14}$$

where q_i is the quantity of electrical charge of the i-th node. The approximate preconditioning matrix on single horizontal direction will be

$$\tilde{\boldsymbol{H}}_{f_{\boldsymbol{x},\boldsymbol{x}}} = \text{diag}\left(\sum_{e\in E_1} w_e + \lambda q_1, \cdots, \sum_{e\in E_n} w_e + \lambda q_n\right). \tag{15}$$

When net weights are all equal to 1, $\sum_{e\in E_i} w_e$ will be degraded to $|E_i|$ which stands for the total number of nets incident to the i-th node. Together with the vertical direction, the preconditioned gradient vector will be $\nabla f_{\text{precond}} = \tilde{\boldsymbol{H}}_f^{-1}\nabla f$.

IV. Experimental Results

A. Experimental Setup

We conduct the experiments on the ICCAD 2015 contest benchmark suites [23]. TABLE I shows the parameters of the circuit designs. All the cases are relatively large and most of them contain millions of cells and nets. No movable macros are

TABLE I Statistics of the ICCAD2015 contest benchmarks [23].

case name	#cells	#nets	#pins	#rows
superblue1	1209716	1215710	3767494	1829
superblue3	1213253	1224979	3905321	1840
superblue4	795645	802513	2497940	1840
superblue5	1086888	1100825	3246878	2528
superblue7	1931639	1933945	6372094	3163
superblue10	1876103	1898119	5560506	3437
superblue16	981559	999902	3013268	1788
superblue18	768068	771542	2559143	1788

included in the benchmark suites. Our algorithm is implemented in C++ based on the open-source placer DREAMPlace [24] and the open-source timer OpenTimer [32]. Remarkably, we manage to make full use of GPU resources in both core placement [24] and timing analysis [33]. For a fair comparison, we follow the exact default hyperparameter settings of DREAMPlace [24].

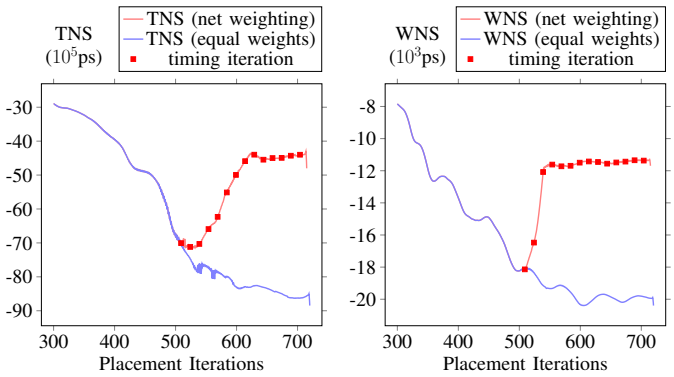
B. TNS and WNS Improvement

It is important to determine when we should set up observation, perform timing analysis and update net weights. it is impossible to perform timing analysis at each iteration, as it will introduce huge overhead. Empirically, cell locations at the earlier stages are highly overlapped, and thus unreliable for timing analysis. A possibly appropriate time to perform timing analysis is when the cells are roughly even out by density forces. In our experiments, we evaluate timing metrics and update net weights every 15 iterations after the 500th iteration of the global placement. Additionally, we use hyperparameters manually customized in Equation (11) to update net weights. We use the evaluation script provided by the ICCAD 2015 contest to evaluate our placement result. The results are listed in TABLE II. All the results are evaluated after Abacus legalization [34]. As shown in the table, we can achieve a significant improvement on both TNS (46.83% on average) and WNS (30.27% on average), compared to the DREAMPlace [24] without any timing-aware optimization.

Also, we implement the classic dynamic net weighting scheme in [9]. Originally, this net weighting scheme is designed for timing-driven quadratic placement. We integrate the net weighting part for timing optimization into our implementation and make a comparison. The results are listed in the second column of TABLE II. We use **boldface** to emphasize the best one among the three results, and color the second one with brown. As shown in the table, our net weighting scheme can outperform [9] a lot on TNS. This result brings us very positive enlightenment that it is absolutely useful to consider timing-aware optimization at the global placement stage. Both TNS and WNS can be improved a lot compared to DREAMPlace without any timing optimization.

C. Visualization

To visualize the impact of net weighting on TNS and WNS, we take superblue18 as an example and plot the TNS and WNS values after the 300th iteration in Fig. 5. Starting from the 300th iteration, the cells have begun repelling each other, and therefore the wirelength keeps increasing, which also decreases TNS and WNS. The blue curves correspond to the results without timing optimization, while the red curves illustrate how the objectives



(a) The TNS curve in placement (b) The WNS curve in placement

Fig. 5 The TNS and WNS values at each placement iteration after the 300th iteration for superblue18.

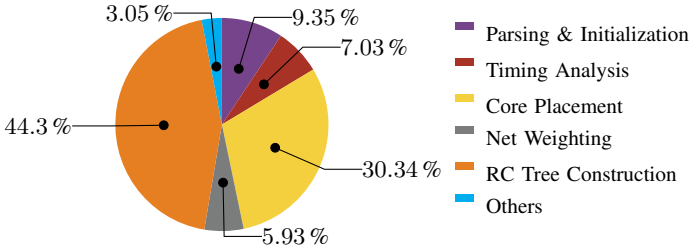


Fig. 6 The runtime breakdown on ICCAD2015 contest benchmark superblue18.

vary with net weighting. We scatter red squares to emphasize the timing iterations.

- At nearly every timing iteration, marked with red color in Fig. 5, TNS can get improved at once, especially when starting to break the balance of net weights.
- WNS will quickly and significantly be optimized after one or two net weighting steps. After that, it almost remains stable during the later stages of global placement.

Providing that our net weighting algorithm works on every net instead of those only on some critical paths at a timing iteration, it is quite reasonable to be effective when optimizing TNS, which may incorporate numerous critical or nearly critical paths. As for WNS, which may only give information about the worst path, it will be quickly optimized when first applying net weighting. At later stages, other critical or nearly critical paths will be taken more into consideration, and that is an important reason why it is hard to further optimize WNS during global placement.

D. Runtime Breakdown

Compared to DREAMPlace [24] which is very powerful to optimize cell locations with GPU-accelerated techniques, timing-driven placement must take extra costs to perform static timing analysis and translate the feedbacks into certain operations. Hence, it is unavoidable to significantly sacrifice runtime performance for timing optimization.

The runtime results are listed in the third column of TABLE II, with column name **RT**. Our weighting scheme is faster than [24] + [9], as we do not need to explicitly extract critical paths, which is very time-consuming. Compared to [24] without any timing-

TABLE II Comparison among DREAMPlace [24], DREAMPlace [24]+ [9], and our algorithm. The best results are emphasized with **boldface**, and the second-best results are colored in brown.

case	DREAMPlace [24]			DREAMPlace [24]+ [9]			Ours		
	TNS (10^5ps)	WNS (10^3ps)	RT (s)	TNS (10^5ps)	WNS (10^3ps)	RT (s)	TNS (10^5ps)	WNS (10^3ps)	RT (s)
superblue1	-252.359	-18.5414	**164.69**	-121.963	**-13.1548**	1320.73	**-85.0315**	-14.1031	977.56
superblue3	-88.4701	-33.2509	**153.95**	-61.2222	**-15.6518**	1247.24	**-54.7427**	-16.4341	952.11
superblue4	-196.498	-21.4654	**112.33**	-177.800	**-11.8600**	910.77	**-144.380**	-12.7808	610.26
superblue5	-208.943	-48.4825	**202.87**	-108.019	-47.7110	1758.97	**-95.7820**	**-26.7602**	1343.46
superblue7	-161.989	-20.3957	**249.32**	-84.3107	-19.9126	1968.70	**-63.8629**	**-15.2163**	1537.42
superblue10	-839.134	-33.7599	**308.81**	-786.359	**-29.0470**	1871.55	**-768.748**	-31.8796	1288.63
superblue16	-438.267	-16.8146	**102.88**	-175.543	-18.5297	875.13	**-124.181**	**-12.1115**	542.15
superblue18	-90.4280	-20.1261	**104.06**	-69.4700	**-11.7831**	887.29	**-47.2458**	-11.8705	657.47
Average Ratio	×2.150	×1.539	×**0.177**	×1.267	×1.167	×1.395	×**1.000**	×**1.000**	×1.000

aware optimization, we roughly take 5 times runtime to optimize negative slacks. Fig. 6 plots the detailed runtime breakdown for superblue18. We are still facing the runtime bottleneck dominated by the RC tree construction. The construction of RC trees is completely accomplished on CPUs and will take a long time, especially for large nets. Considering that timing analysis must be called multiple times to incorporate changes of cell locations, the overhead of RC tree construction and timing analysis should be the main focus for acceleration.

V. Conclusion

In this paper, we propose a momentum-based net weighting scheme for timing-driven global placement and improve the preconditioner accordingly. The evaluation results on ICCAD2015 contest benchmarks show that we can achieve a significant improvement on both TNS and WNS. The results of this paper enlighten us that, although most timing-aware optimization methods are performed at incremental stages, it is still very effective to take consider timing at the earlier stages of physical design, especially global placement.

References

[1] I. L. Markov, J. Hu, and M.-C. Kim, "Progress and challenges in VLSI placement research," *Proceedings of the IEEE*, vol. 103, no. 11, pp. 1985–2003, 2015.

[2] M. Burstein and M. N. Youssef, "Timing influenced layout design," in *Proc. DAC*. IEEE, 1985, pp. 124–130.

[3] A. E. Dunlop, V. D. Agrawal, D. N. Deutsch, M. Jukl, P. Kozak, and M. Wiesel, "Chip layout optimization using critical path weighting," in *Proc. DAC*. IEEE, 1984, pp. 133–136.

[4] H. Chang, E. Shragowitz, J. Liu, H. Youssef, B. Lu, and S. Sutanthavibul, "Net criticality revisited: An effective method to improve timing in physical design," in *ispd*, 2002, pp. 155–160.

[5] T. Kong, "A novel net weighting algorithm for timing-driven placement," in *Proc. ICCAD*, 2002, pp. 172–176.

[6] B. Halpin, C. R. Chen, and N. Sehgal, "A sensitivity based placer for standard cells," in *Proceedings of the 10th Great Lakes symposium on VLSI*, 2000, pp. 193–196.

[7] T.-Y. Wang, J.-L. Tsai, and C. C.-P. Chen, "Sensitivity guided net weighting for placement driven synthesis," in *Proc. ISPD*, 2004, pp. 124–131.

[8] Z. Xiu and R. A. Rutenbar, "Timing-driven placement by grid-warping," in *Proc. DAC*, 2005, pp. 585–591.

[9] H. Eisenmann and F. M. Johannes, "Generic global placement and floorplanning," in *Proc. DAC*, 1998, pp. 269–274.

[10] B. M. Riess and G. G. Ettelt, "Speed: Fast and efficient timing driven placement," in *Proc. ISCAS*, vol. 1. IEEE, 1995, pp. 377–380.

[11] B. Obermeier and F. M. Johannes, "Quadratic placement using an improved timing model," in *Proc. DAC*. IEEE, 2004, pp. 705–710.

[12] A. B. Kahng, J. Lienig, I. L. Markov, and J. Hu, *VLSI physical design: from graph partitioning to timing closure*. Springer Science & Business Media, 2011.

[13] W. K. Luk, "A fast physical constraint generator for timing driven layout," in *Proc. DAC*, 1991, pp. 626–631.

[14] T. Gao, P. M. Vaidya, and C. Liu, "A Performance Driven Macro-Cell Placement Algorithm." in *Proc. DAC*, 1992, pp. 147–152.

[15] R.-S. Tsay and J. Koehl, "An analytic net weighting approach for performance optimization in circuit placement," in *Proc. DAC*, 1991, pp. 620–625.

[16] K. Rajagopal, T. Shaked, Y. Parasuram, T. Cao, A. Chowdhary, and B. Halpin, "Timing driven force directed placement with physical net constraints," in *ispd*, 2003, pp. 60–66.

[17] A. Chowdhary, K. Rajagopal, S. Venkatesan, T. Cao, V. Tiourin, Y. Parasuram, and B. Halpin, "How accurately can we model timing in a placement engine?" in *Proc. DAC*, 2005, pp. 801–806.

[18] M. A. Jackson and E. S. Kuh, "Performance-driven placement of cell based IC's," in *Proc. DAC*, 1989, pp. 370–375.

[19] W. Swartz and C. Sechen, "Timing driven placement for large standard cell circuits," in *Proc. DAC*, 1995, pp. 211–215.

[20] T. Hamada, C.-K. Cheng, and P. M. Chau, "Prime: A timing-driven placement tool using a piecewise linear resistive network approach," in *Proc. DAC*, 1993, pp. 531–536.

[21] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," vol. 323, no. 6088, pp. 533–536, 1986.

[22] J. Lu, P. Chen, C.-C. Chang, L. Sha, D. J.-H. Huang, C.-C. Teng, and C.-K. Cheng, "ePlace: Electrostatics-based placement using fast fourier transform and Nesterov's method," vol. 20, no. 2, pp. 1–34, 2015.

[23] M.-C. Kim, J. Hu, J. Li, and N. Viswanathan, "ICCAD-2015 CAD contest in incremental timing-driven placement and benchmark suite," in *Proc. ICCAD*. IEEE, 2015, pp. 921–926.

[24] Y. Lin, Z. Jiang, J. Gu, W. Li, S. Dhar, H. Ren, B. Khailany, and D. Z. Pan, "Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement," *IEEE TCAD*, 2020.

[25] M.-K. Hsu, Y.-W. Chang, and V. Balabanov, "TSV-aware analytical placement for 3D IC designs," in *Proc. DAC*, 2011, pp. 664–669.

[26] M.-K. Hsu, V. Balabanov, and Y.-W. Chang, "TSV-aware analytical placement for 3-D IC designs based on a novel weighted-average wirelength model," *IEEE TCAD*, vol. 32, no. 4, pp. 497–509, 2013.

[27] N. Maheshwari and S. S. Sapatnekar, "Timing Analysis of Sequential Circuits," in *Timing Analysis and Optimization of Sequential Circuits*. Springer, 1999, pp. 7–31.

[28] R. B. Hitchcock, "Timing verification and the timing analysis program," in *Proc. DAC*. IEEE, 1982, pp. 594–604.

[29] D. Z. Pan, B. Halpin, and H. Ren, "21 Timing-Driven Placement," *Handbook of Algorithms for Physical Design Automation*, p. 423, 2008.

[30] C. Chu and Y.-C. Wong, "FLUTE: Fast lookup table based rectilinear Steiner minimal tree algorithm for VLSI design," *IEEE TCAD*, vol. 27, no. 1, pp. 70–83, 2007.

[31] W. C. Elmore, "The transient response of damped linear networks with particular regard to wideband amplifiers," *Journal of applied physics*, vol. 19, no. 1, pp. 55–63, 1948.

[32] T.-W. Huang and M. D. Wong, "OpenTimer: A high-performance timing analysis tool," in *Proc. ICCAD*, 2015, pp. 895–902.

[33] Z. Guo, T.-W. Huang, and Y. Lin, "Gpu-accelerated static timing analysis," in *Proc. ICCAD*, 2020, pp. 1–9.

[34] P. Spindler, U. Schlichtmann, and F. M. Johannes, "Abacus: Fast legalization of standard cell circuits with minimal movement," in *ispd*, 2008, pp. 47–53.