Timing Driven Placement

David Z. Pan, Bill Halpin, Haoxing Ren

1 Introduction

The placement algorithms presented in the previous chapters mostly focus on minimizing the total wirelength (TWL). Timing-driven placement (TDP) is designed specifically targeting wires on timing critical paths. It shall be noted that a cell is usually connected with two or more cells. Making some targeted nets shorter during placement may sacrifice the wirelengths of other nets that are connected through common cells. While the delay on critical paths decreases, other paths may become critical. Therefore, timing-driven placement has to be performed in a very careful and balanced manner.

Timing-driven placement has been studied extensively over the last two decades. The drive for new methods in timing-driven placement to maximize circuit performance is from multiple facets due to the technology scaling and integration: (1) growing interconnect versus gate delay ratios, (2) higher levels of on-die functional integration - which makes global interconnects even longer, (3) increasing chip operating frequencies which make timing closure tough, (4) increasing number of macros and standard cells for modern system-on-chip (SOC) designs. These factors create continuing challenges to better timing-driven placement

Timing-driven placement can be performed at both global and detailed placement stages (see previous chapters on placement). Historically, timing-driven placement algorithms can be roughly grouped into two classes: *net-based* and *path-based*. The net-based approach deals with nets only, with the hope that if we handle the *nets* on the critical paths well, the entire critical *path* delay may be optimized *implicitly*. The two basic techniques for net-based optimization are through *net weighting* [17, 5, 37, 50] and *net constraints* [22] [63] [55] [48] [24] [29]. The path-based approach *directly* works on all or a subset of paths [31, 59, 61, 11]. The majority path-based approaches formulate the problem into a mathematical programming framework (e.g., linear programming). There are pros and cons for both net and path based approaches in terms of runtime/scalability, ease of implementation, controllability, and so on. Modern timing driven placement techniques tend to use some hybrid manner of both net-based and path-based approaches [39].

In this chapter, we will discuss fundamental algorithms as well as recent trends of timing-driven placement. Due to the large amount of works in timing-driven placement, it is not possible to exhaust all of them in this chapter. Instead, we will describe the basic ideas and fundamental techniques, and point out recent researches and possible future directions. We will first cover the basic building blocks for timing driven placement. Then the next two sections will discuss net-based approaches, i.e., through net weighting and net constraints. Then we will survey the basic formulations and algorithms behind the path (or timing graph) based approach. Additional techniques and issues in the context of timing driven placement will be discussed, followed by conclusions.

2 Building Blocks and Classification

2.1 Net Modeling

Given a placement, net modeling answers a fundamental question how the net is modeled for its routing topology and wirelength computation/estimation. Figure 1 shows different net modeling strategies for a

multiple-pin net.

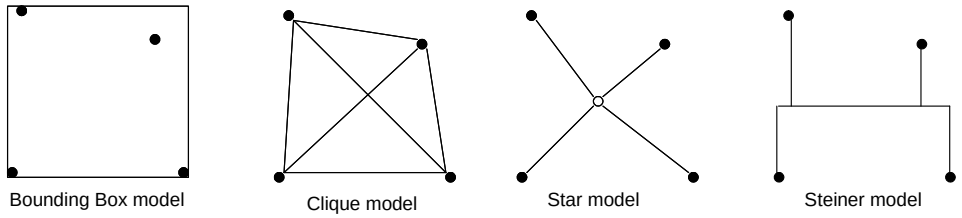

Figure 1: Different net models which can be used for placement.

The simplest and most widely used method to compute wirelength is the half-perimeter wirelength (HPWL) of its bounding box. For a net i, let l_i, r_i, u_i and b_i represent the left, right, top, and bottom locations of its bounding box. Then the HPWL of net i is

$$HPWL_i = r_i - l_i + u_i - b_i \tag{1}$$

HPWL is the lower bound for wirelength estimation, and it is accurate for two and three-pin nets, which account for the majority nets.

In analytical placement engines, wirelength is often modeled as a quadratic term (or pseudo-linear term as in recent literature [34] [6]). In those engines, clique and star models are often used. In the clique model, an edge is introduced between every pin pair of the net. In the star model, an extra star point located at the geometric center is created and each pin is connected to the star point. In general, small nets (e.g., less than 5 pins) can use the clique model, but for large nets with a lot of pins, clique model is not friendly to the matrix solvers since it creates dense matrices. Star models are preferred for large nets. For the clique model, since it is a complete graph with far more edges than necessary to connect the net, each edge is usually assigned a weight of $2/n$ (where n is the number of pins of the net) [4].

The *bounding box*, *clique*, and *star* models are three most popular net models. There are other net models, such as Steiner trees, which are more accurate for nets with four or more pins. However, in most designs two and three pin nets are the majority of the entire netlist. For example, in the industry circuit suite from [3], two and three pin nets constitute 64% and 20% of the total nets, respectively [56]. With exception of very few placers [1], Steiner tree based models are seldom used since they are computationally expensive. There are some recent works trying to link Steiner tree with placement, e.g., in a partition-based placer [53]. More research is needed to evaluate or make Steiner-based placement mainstream.

2.2 Timing Analysis and Metrics

As its name implies, timing driven placement has to be guided by some timing metrics, which in turn need delay modeling and timing analysis. Timing-driven placement algorithms can use different levels of timing models to tradeoff accuracy and runtime. In general, the switch level RC model for gates and Elmore delay model for interconnects are fairly sufficient. There are more accurate models [47], but they are not extensively used in placement. One main reason is the higher runtime. The other reason is that during placement, routing is not done yet. It is not very meaningful to use more accurate models if errors from those uncertainties are even greater.

Based on the gate and interconnect delay models, static or even path[1] based timing analysis can be performed. Static timing analysis [40] computes circuit path delays using the critical path method [52]. From the set of *arrival times (Arr)* asserted on timing starting points and *required arrival times (Req)* asserted on timing-end points, static timing analysis propagates (latest) arrival time forward and (earliest) required

[1] In most cases, static timing analysis is sufficient for timing driven placement. The path-based timing analysis is more accurate, e.g., to capture false paths. But it is very time consuming, and one may do it only if necessary, e.g., on a set of critical paths.

arrival time backward, in the topological order. Then the slack at any timing point t is the difference of its required arrival time minus its arrival time.

$$Slk(t) = Req(t) - Arr(t) \tag{2}$$

STA can be performed incrementally if small changes in the netlist are made. For more details on delay modeling and timing analysis, the reader is referred to Chapter 3.

Timing convergence metrics measure the extent to which a placement satisfies timing constraints. They also give an indication of how difficult it would be for a design engineer to manually fix timing problems. The most commonly used timing closure metric is the *worst negative slack* (WNS)

$$WNS = \min_{t \in P_o} Slk(t) \tag{3}$$

where P_o is the set of timing end points, i.e., primary outputs (POs) and data inputs of memory elements. To achieve timing closure, WNS should be non-negative. For nanometer designs with growing variability, one may set the slack target to be a positive value to safe guard variations from process, voltage, or thermal issues. The WNS, however, only gives information about the worst path. It is possible that two placement solutions have similar WNS values, but one has only a single critical path while the other has thousands of critical and near critical paths. The *figure of merit* (FOM) is another very important timing closure metric [50]. It can be defined as follows.

$$FOM = \sum_{t \in P_o, Slk(t) < Slk_t} (Slk(t) - Slk_t) \tag{4}$$

where Slk_t is the slack target for the entire design. If $Slk_t = 0$, the FOM is reduced to the *total negative slack* (TNS) [48].

2.3 Overview of Timing-Driven Placement

The overview of timing driven placement is shown in Figure 2. It has three basic components: timing analysis, core placement algorithms, and interfaces between them by translating timing analysis/metrics into certain weights or constraints for core placement engines to drive and guide timing driven placement.

The previous section discusses the basics of timing analysis and metrics from a given netlist. However, which netlist to start with so that we can have a meaningful timing analysis to guide timing-driven placement is a very important yet open question. For example, shall we start from an unplaced netlist or some initial placement? For modern timing closure, many buffers also need to be inserted for high fanout nets and long interconnects to get a reasonable timing picture (otherwise, there may be many loading/slew violations which make timing reports meaningless). On the other hand, those buffers will change the netlist structure for timing-driven placement. Shall they be kept or stripped out during timing driven placement? There is very little literature covering this netlist preparation step. A reasonable strategy can be as follows. First, we start with some initial placement (e.g., wirelength driven), then perform some rough buffering/fanout optimization to get a reasonable timing estimation for the entire chip to guide timing driven placement engine. Whether to keep those buffers during timing driven placement may vary among different physical synthesis systems.

Placement has been one of the most heavily studied physical design topics. Some of the most popular placement algorithms include analytical/force-directed placement, partition-based placement, simulated annealing based placement, and linear-programming based placement. The reader is referred to Chapters 15-18 for detailed discussions.

The most interesting aspect of the timing driven placement is the mechanism to translate timing metrics into actions to drive the core place engines. The focus of the rest of this chapter will be on this aspect.

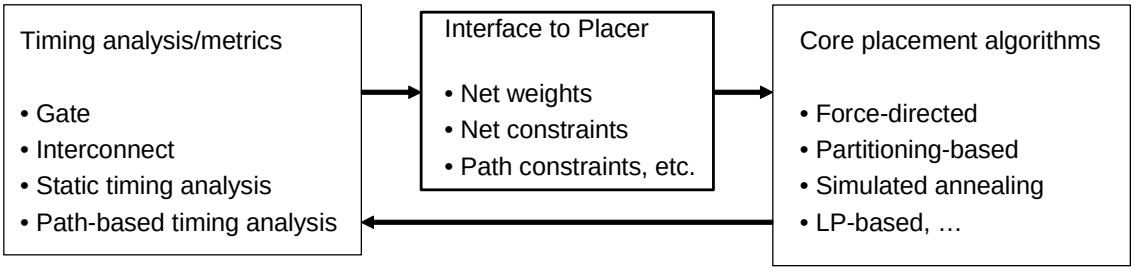

Figure 2: Basic building blocks and overview of timing driven placement.

Based on that, the timing-driven placement can be roughly classified into *net-based* and *path-based* approaches. The net-based approach, as its name implies, deals with individual nets. Since timing analysis inherently deals with paths (with timing propagation), the timing information is then translated into either *net constraints* or *net weights* [17, 5, 37, 50] to guide timing-driven placement engines. The main idea of net constraint generation (or delay budgeting) is to distribute slack for each path to its constituent nets such that a zero-slack solution is obtained. The delay budget for each net is then translated into its wirelength constraint during placement. The main idea of net weighting is to assign higher net weights to more timing-critical nets while minimizing the total weighted wirelength objective. Net weighting gives direction for timing optimization through shortening critical nets, but it does not have exact control since the objective is the total weighted wirelength; the net constraint approach specifies that, but it may be too much constrained in terms of global optimization. Both processes can be iteratively refined as more accurate timing information is obtained during placement. A systematic way of explicit perturbation control is important for net-based algorithms.

The path-based approach directly works on all or a subset of paths. The majority of this approach formulate the problem into mathematical programming framework (e.g., linear programming). It usually maintains an accurate timing view during the placement optimization [7]. However, its drawback is its poor scalability and complexity due to possible exponential number of paths to be simultaneously optimized [7]. An effective technique is to embed *timing graph/constraint* through auxiliary variables [31]. The mathematical programming based approach needs to deal with cell overlapping issues, e.g., through partitioning. The path-based timing can also be evaluated in a simulated annealing framework [61].

Both net-based and path-based approaches have pros and cons. Path-based in general has more accurate timing view and control, but it suffers from poor scalability. The net-based approaches, in particular net weighting, have low computational complexity and high flexibility. Thus they are suitable for large ASIC/SOC designs with millions of placeable objects. Recent research shows that hybrid of these two basic approaches are promising [39].

3 Net Weighting Based Approach

Classic placement algorithms optimize the total wirelength. They can be easily modified to be timing-driven using the net weighting technique, which assigns different weights to different nets such that the placer minimizes the total weighted wirelength (if all the weights are the same, it degenerates into the classic wirelength-driven placement). Intuitively, a proper net weighting should assign higher weights on more timing-critical nets, with the hope that the placement engine will reduce the lengths of these critical nets and thus their delays to achieve better overall timing.

Net weighting based timing-driven placement is very simple to implement and less computational intensive. As modern VLSI designs have millions placeable objects (gates/cells/macros), net weighting is attractive because of its simplicity. Almost all placement algorithms support net weighting. Quadratic placement can optimize the weighted quadratic wirelength, partition-based placement can optimize the weighted cut

size (see Chapter 8), and simulated annealing based placement can optimize the weighted linear wirelength, etc.

While net weighting appears to be straightforward, it is not easy to generate a good net weighting. Higher net weights on a set of critical nets in general shall reduce their wirelengths and delays, but other nets may become longer and more critical. In this section, we will review two basic sets of net weighting algorithms: static net weighting and dynamical net weighting. Static net weighting assigns weights once before timing-driven placement and the weights do not change *during* timing-driven placement. Dynamic net weighting updates weights during the timing-driven placement process.

3.1 Static Net Weighting

Static net weighting computes the net weights once before timing driven placement. It can be divided into two categories: empirical net weighting and sensitivity based net weighting. Empirical net weighting methods compute weights based on certain criticality factors, such as slack, cycle time, and fanout. Critical nets are assigned higher net weights. Sensitivity based net weighting computes weights based on the sensitivity analysis of net weights to factors such as WNS and FOM. The key difference of these two net weighting schemes is that sensitivity based approach has some look-ahead mechanism that can estimate the impact of net weighting on key factors. Therefore it assigns higher net weights on those nets that have bigger impacts on the overall timing closure goal.

3.1.1 Slack-Based Net Weighting

Empirical net weighting assigns net weight based on the criticality of the net, which indicates how much the placer should reduce the wirelength on this net. The criticality computation can be from the static timing analysis (STA). Assuming there is only one clock period, the net criticality can often be measured by slack. Nets with negative slacks are critical nets and are assigned higher net weights than those nets with positive slacks.

$$w = \begin{cases} W_1 & \text{slack} < 0 \\ W_2 & \text{slack} \geq 0 \end{cases} \tag{5}$$

where W_1 and W_2 are positive constants and $W_1 > W_2$. Among the critical nets that have negative slacks, higher weights can also be given to those which are more negative. One can either define a continuous model [17] or a step-wise model [5] to compute weights based on the slack distribution, as shown in Figure 3.

It shall be noted that net weights shall not continue to increase when slack is less than certain threshold. This is because slack can be very negative due to invalid timing assertions. Usually placers do not need very high net weights to pull nets together.

For some placers, one might add an exponential component into net weighting [2] to further emphasize critical nets.

$$w = (1 - \frac{slack}{T})^{\alpha} \tag{6}$$

where T is the longest path delay and α is the criticality exponent. If there are multiple clocks in the design, the clock cycle time can be considered during net weighting. Nets on paths of shorter cycle time should have higher weights than those of longer cycle time with the same slack. For example, we can use T is (6) to represent different cycle times. Paths of long cycle time clocks get a larger T than those of short cycle clocks.

$$w = (1 - \frac{slack}{T_{clk}})^{\alpha} \tag{7}$$

where T_{clk} is the clock cycle time for a particular net.

There are other empirical factors that can be considered for net weighting, e.g., path depth and driver strength [41]. A deep path, which has many stages of logic, is more likely to have longer wire length. The

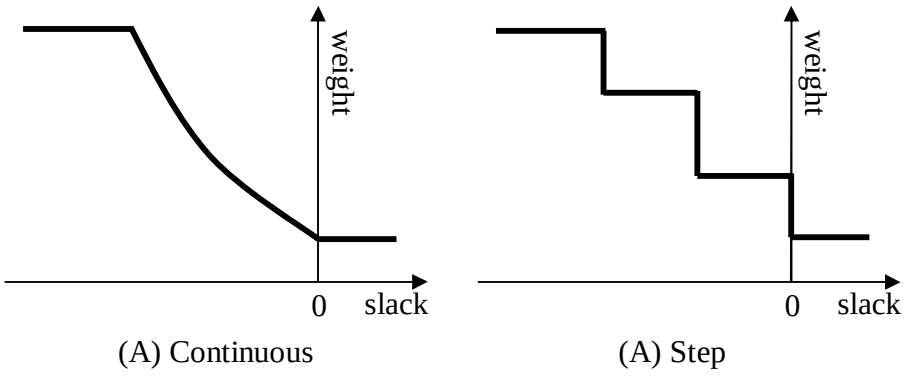

Figure 3: Net weight assignment based on slack using continuous or step model.

end points of this path could be placed far away from each other, therefore worse timing is expected than a path with less number of stages. A net with a weak driver would have longer delay than a strong driver with the same wire length. Therefore, net weight should be proportional to the longest path depth (which can be computed by running breadth first search twice: once from PIs and second time from POs), and inversely to slack and driver strength, i.e.,

$$w \approx D_l \times R_d \tag{8}$$

where D_l is the longest path depth, and R_d is the driver resistance. A weaker driver has a larger effective driving resistance, thus the net it drives will have a bigger net weight.

One could also consider path sharing during net weighting. Intuitively a net on many critical paths should be assigned a higher weight because reducing the length of such net can reduce the delay on many critical paths.

$$w \approx slack \times GP \tag{9}$$

where GP is the number of critical paths passing this net. Suppose we assign two variables for each net p on the timing graph: $F(p)$ the number of different critical paths starting from timing beginning point (PI) to net p; and $B(p)$ the number of different critical paths from net p to timing end points (PO). The total number of critical paths passing through net p is then $GP(p) = F(p) \times B(p)$. This net weighting assignment only considers the sharing effect of critical paths, and each path has the same impact of the net weight. Kong proposed an accurate, all path counting algorithm PATH [37], which considers both noncritical and critical paths during path counting. It can properly scale the impact of all paths by their relative timing criticalities.

To perform net weighting for unplaced designs, STA can use the wire load model, e.g., based on fanout, to estimate the delay (compared to placed designs, STA can use the actual wire load to compute delay). Normally it is not accurate with wire load models. Therefore, for an unplaced design, an alternative way of generating weights is to use fanout and delay bound [8] instead of slack. Fanout is used to estimate wirelength and wire delay [38], and delay bound is the estimated allowable wire delay, i.e., any wire delay above this bound would result in negative slack. The weight can be computed as the ratio of fanout and delay bound.

$$w \approx \frac{fanout}{net\ delay\ bound} \tag{10}$$

In general, as the impact of net weight assignment is not very predictable, extensive parameter tuning may be needed to make it work on specific design styles.

3.1.2 Sensitivity Based Net Weighting

Net weighting can help improve timing on critical paths. However, it may have negative effects on total wirelength. Assigning higher net weights on too many nets may result in significant degradation of wire-

length, thus may introduce routing congestion and new critical paths. To apply high net weights only on those nets which will result in large gain in timing, we can reduce wirelength degradation and other side effects of net weighting.

Sensitivity based net weighting tries to predict the net weighting impact on timing and use that sensitivity to guide net weighting [50] [25]. The question that sensitivity analysis tries to address is: given an initial placement from an initial net weighting scheme, if we increase the weight for a net i by certain nominal amount, how much improvement net i will get for its worst slack (WNS) and the overall figure of merit, FOM (or in a more familiar term, total negative slack - TNS when the slack threshold for FOM is zero). With detailed sensitivity analysis, larger weights could be assigned to a net whose weight change can have a larger impact to delay. In this section, we will explain how to estimate both slack sensitivity and TNS sensitivity to net weights and how to use those sensitivities to compute net weights.

First, one needs to estimate the impact of net weight change to wirelength, i.e. the wirelength sensitivity to net weight. This sensitivity depends on the characteristics of a placer. It is not easy to estimate such sensitivity for min-cut or simulated annealing based algorithms. But for quadratic placement, one can come up with an analytical model to estimate it. Based on Tsay's analytical model [63], the wirelength sensitivity to net weight can be derived [50] as

$$S_W^L(i) = \frac{\Delta L(i)}{\Delta W(i)} = -L(i) \cdot \frac{W_{src}(i) + W_{sink}(i) - 2W(i)}{W_{src}(i)W_{sink}(i)} \tag{11}$$

where $L(i)$ is the initial wire length of net i, $W(i)$ is the initial weight of net i, $W_{src}(i)$ is the total initial weight on the driver/source of net i (simply the summation of all nets that intersect with the driver), and $W_{sink}(i)$ is the total initial weight on the receiver/sink of net i. Intuitively, (11) implies that if the initial wire length $L(i)$ is longer, for the same amount of nominal weight change, it expects to see bigger wire length change. Meanwhile, if the initial weight $W(i)$ is relatively small, its expected wire length change will be bigger. The negative sign means that increasing netweight will reduce wirelength.

The next step is to estimate the wirelength impact on delay. Using the switch level RC device model and the Elmore delay model [13], the delay sensitivity to wirelength can be estimated as:

$$S_L^T(i) = \frac{\Delta T(i)}{\Delta L(i)} = rcL(i) + cR_d + rC_l \tag{12}$$

where r and c are the unit length wire resistance and capacitance, respectively. R_d is the output resistance of the net driver, and C_l is the load capacitance. It implies that for a given technology (fixed r and c), the delay of a long wire with a weak driver and large load will be more sensitive to the same amount of wire length change.

With wirelength sensitivity and delay sensitivity, one can compute the slack sensitivity to net weight as:

$$S_W^{Slk}(i) = \frac{\Delta Slk(i)}{\Delta W(i)} = -\frac{\Delta T(i)}{\Delta L(i)} \cdot \frac{\Delta L(i)}{\Delta W(i)} = -S_L^T(i)S_W^L(i) \tag{13}$$

Total negative slack (TNS) is an important timing closure objective. The TNS sensitivity to net weight is defined as follows:

$$S_W^{TNS}(i) = \Delta TNS/\Delta W(i) \tag{14}$$

Note that TNS improvement comes from the delay improvement of this net, equation (14) can be decomposed into:

$$S_W^{TNS}(i) = \frac{\Delta TNS}{\Delta T(i)} \frac{\Delta T(i)}{\Delta W(i)} = -K(i)S_W^{Slk}(i) \tag{15}$$

where $K(i) = \frac{\Delta TNS}{\Delta T(i)}$, which means how much TNS improvement it can achieve by reduce net delay $T(i)$. It has been shown in [50] that $K(i)$ is equal to the negative of the number of critical timing end points whose

Algorithm 1 Counting the number of influenced timing critical end points for each net

1: decompose nets with multiple sink pins into sets of driver-to-sink nets
2: initialize $K(i) = 0$ for all nets and timing points
3: sort all nets in topological order from timing end points to timing start points
4: **for all** P_o pin t **do**
5: set $K(t)$ to be 1 if t is timing critical (i.e., $Slk(t) < Slk_t$; otherwise set $K(t)$ to be 0
6: **for all** net i in the above topologically sorted order **do**
7: **for all** sink pin j of net i **do**
8: $K(i) = K(i) + K(j)$
9: propagate $K(i)$ of net i to its driver input pins: only the most critical input pin gets $K(i)$; other pins will have $K = 0$ because they are not on the critical path of net i, thus cannot influence the timing end points from net i

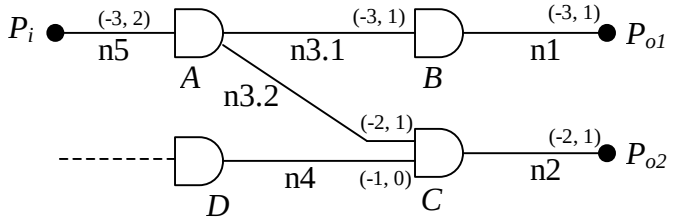

Figure 4: Counting the number of influenced timing end points

slacks are influenced by net i with a nominal $\Delta T(i)$ and can be computed efficiently as shown in following algorithm..

As an example, Figure 4 shows two paths from a timing begin point P_i to timing end points P_{o1} and P_{o2}. Net $n3.1$ and $n3.2$ are the decomposed driver-to-sink nets from the original net $n3$. The pairs in the figure such as "(-3, 1)" have the following meaning: the first number is the slack, and the second number is the K value. Since the slacks at P_{o1} and P_{o2} are -3ns and -2ns, respectively (worse than the slack target of 0), the K values for P_{o1} and P_{o2} are both 1. We can see how the K values are propagated from PO to PI. Note that for gate C, the upper input pin has slack of -2ns while the lower input pin has slack of -1ns, thus the upper pin is the most timing critical pin to gate C and it will influence the slack of P_{o2}. The lower pin of C does not influence P_{o2}.

The sensitivity based net weighting scheme starts from a set of initial net weights (e.g., uniform netweighting at the beginning), and computes a new set of net weights that would maximize the slack and TNS gain. Since the sensitivity analysis works best when the net weights are updated in small steps from their initial values, it also adds a constant of total change to bound the net weights. The net weight can be computed as:

$$W(i) = \begin{cases} W_{org}(i) & Slk(i) > 0 \\ W_{org}(i) + \alpha[Slk_t - Slk(i)]S_W^{Slk}(i) + \beta S_W^{TNS}(i) & Slk(i) \leq 0 \end{cases} \quad (16)$$

where $W_{org}(i)$ is the original net weight, α and β set the bound of net weight changes, and control the balance between worst negative slack and total negative slack.

3.2 Dynamic Net Weighting

Static net weighting computes net weights once and does not update them during timing driven placement. However, wirelengths change during and after placement, and the original timing analysis may not be valid. To overcome this problem, dynamical net weighting methods were proposed to adjust weights during placement based on timing information available at current placement stage.

A simple dynamical net weighting scheme is to run multiple placement and net weighting iterations. This scheme can be applied on any placement and net weighting algorithms. This simple scheme, however, is often hard to converge without careful net weighting assignment. This is the so-called *oscillation* problem [12]. Weights are assigned by performing timing analysis for some given placement solution at the *n*-th iteration [41]. Critical nets receive higher weights. At next iteration, the lengths of those critical nets are reduced, while the lengths of some noncritical nets may be increased, resulting in a different set of critical & noncritical nets. If a net alternates between critical and noncritical net, we have an *oscillation* problem. To mitigate this problem, one needs to either periodically recompute timing during the placement process [2] [61] or use historical net weighting information to achieve *stability* [51] [18].

3.2.1 Incremental Timing Analysis

To periodically update weights during placement, one needs to recompute timing during placement. One could incrementally update timing like [5], which only computes the incremental slack caused by wirelength increments using delay sensitivity to wirelength.

$$s^k(n) = s^{k-1}(n) - \Delta d^k(n) = s^{k-1}(n) - S_L^T(n)\Delta l(n) \tag{17}$$

where $s^k(n)$ is the estimated slack for net n at the k step, $s^{k-1}(n)$ is the slack at $k-1$ step, $\Delta d^k(n)$ is the delay change on net n and S_L^T is the delay to wirelength sensitivity and $\Delta l(n)$ is the wirelength increment.

Using sensitivity analysis can provide a fast estimation for incremental timing analysis. One can also perform a more accurate incremental timing analysis. For example, [51] uses a star net model for placement and netlist changes. The main advantage of this model is that it can calculate individual delay between the source pin and every sink pin of star net more accurately. Assuming given gate coordinates, the star net node is computed as the center of gravity of all pins of the net, and the lengths of all arcs in x and y directions can be obtained. These lengths are used to compute the equivalent lumped elements as used in the derived electrical model. Note that one normally does not perform a full blown static timing analysis during placement, which would do false path detection, early-late mode analysis, etc.

3.2.2 Incremental Net Weighting

To make placement stable with updated weights, we can make use of the historical weights, so called *incremental net weighting*. Different from static net weighting, this method relies on iterations to get the appropriate weights and drives the placement engine along that way.

There are two such algorithms in published literature. One only makes use of the history data of the previous step, the other uses the previous two steps.

In [18], at each step, it first computes the criticality for a net i as

$$c_i^k = \begin{cases} (c_j^{k-1} + 1)/2 & \text{if net } i \text{ is among the 3\% most critical nets} \\ c_j^{k-1}/2 & \text{otherwise} \end{cases} \tag{18}$$

The criticality describes how critical a net tends to be in general. For example, if a net was never critical its criticality is zero whereas an always critical net has a criticality of one. This scheme effectively reduces oscillations of weights.

Once the criticality is computed, the net weight then can be updated as:

$$w_i^k = w_i^{k-1} \times (1 + c_i^k) \tag{19}$$

Therefore, the net with criticality one will have its weight doubled at every iteration, while noncritical net weights will stay the same.

The other net weighting scheme uses the criticality information from the previous two steps [51]. In this approach, the criticality number is simplified to either one or zero. Nets on critical paths get one, while nets on noncritical paths get zero. The net weight is updated as follows.

$$w_i^k = \begin{cases} w^{k-1} + W & \text{if } c_i^k = 1 \\ 1 & \text{if } c_i^k = 0 \wedge c_i^{k-1} = 0 \wedge c_i^{k-2} = 0 \\ \lceil w_i^{k-1}/2 \rceil & \text{if } c_i^k = 0 \wedge c_i^{k-1} = 0 \wedge c_i^{k-2} = 1 \\ w^{k-1} & \text{if } c_i^k = 0 \wedge c_i^{k-1} = 1 \end{cases} \tag{20}$$

In this case, the minimum net weight is 1. If the current criticality is 1, its net weight will be increased by W (> 1), which determines how fast the weight would increase due to criticality. Using the number of pins of a net to set W is a reasonable choice because delays of nets with high fanouts are usually larger and more likely to be critical. If the current step net criticality is 0, the net weight may change depending on the criticalities of the previous two steps.

3.2.3 Placement Implementation

Dynamical net weighting algorithms can be applied to most placement algorithms, e.g., partition based placement [5] [28] [46], quadratic placement [51], and force-directed placement [18].

The implementation of dynamical net weighting on quadratic and force directed placement can be straightforward. Since both placement algorithms provide intermediate gate coordinates at each step, it is easy to estimate wire loads and timing based on those gate coordinates. It is also effective to use the incremental net weighting methods such as (19) and (20) to drive those placement engines because the matrix solvers for those placers usually respond well to weights changes.

For pure partitioning based placement, one can also use similar method, i.e. update weights between each partitioning step [5] [28]. However, the timing analysis in general is not as accurate because partitioning-based placement does not assign exact gate coordinates inside a partition. Thus the weights may not effectively control the partitioning process which aims at minimizing the number of weighted crossings, but not wirelength directly.

One can enforce some cutting constraints to the partitioning algorithm, e.g., the maximum number of times a path can be cut during the iterative partitioning steps [45]. For partitioning based placement, controlling the cut number on paths in addition to weights helps reduce the wirelength on critical nets more efficiently. It is also a dynamic net weighting approach in that it updates the timing criticality during partitioning process and recomputes weight as well. Unlike previous timing analysis methods which recalculate timing based on gate coordinates, it estimates the critical path by the number of cuts a path has been cut during partitioning. Starting from an initial set of most critical nets, it adds some number of critical nets that has been cut to this set. All the critical nets will be limited to be cut only a maximum number of times by setting a higher weight that is equal to the summation of the weights of non-critical nets in a partition, which guarantees that critical nets will not be cut.

In [32], the minimization of the maximal path delay problem is formulated in the min-max, top down partition-based placement for timing optimization. The main technique is the iterative net re-weighting. It another work [33], the concept of boosting factors is introduced which adjusts net weights according to net spans, so that the quadratic wirelength can be reduced. The method skews the netlength distribution produced by a min-cut placer so as to decrease the number of long nets, with minimal impact on the overall wirelength.

4 Net Constraint Based Approach

4.1 Net Constraint Generation

Since interconnect delay is predominately determined by its net length, a natural choice for controlling delay is through *net length constraint* (*NLC*), which limits the maximum length of a net. The net constraint based approach is another popular net-based "interface" between timing analysis and placement to drive the timing-driven placement. The net constraint approach have several attractive qualities compared to the common net weighting approach. It is not possible to predict the exact timing response to a net weight. Since many nets may have weight changes, there may be conflicts with each other. Sometimes it is not even certain that the length of a net will be reduced if it is given a higher net weight. Net constraint approach has more accurate control. The problem then is how to generate a good set of net constraints which are not overly constrained to limit solution space. A common combined flow may be combining netweighting and net constraints, e.g., having netweighting to guide global timing-driven placement and net-constraint generation for incremental/iterative improvement.

The two main steps of net constraint driven placement are:

1. to generate an effective set of *NLC* bounds, and
2. create placers which meet, or nearly meet these bounds.

The following two sections will explore these two net constraint driven goals.

4.1.1 Generating Effective *NLC*s

Many techniques have been proposed for generating *NLC*s and many are similar with the approaches for creating net weights. Many of the original methods attempted to create, in a "single shot", a set of *NLC*s, which when met would result in a design which meets timing requirements. More recently, several works have suggested that *NLC*s should be generated so that the design's target frequency is incrementally improved. The single shot approaches are described first.

4.1.2 "Single Shot" *NLC* Generation

The goal of "single shot" *NLC* generation is to perform a slack budgeting giving timing constraint for each net, which when realized will meet the timing frequency goal. These timing budgets are then used to generate a physical bound for the *NLC* using silicon process parasitic parameters.

In [42], the zero-slack algorithm (ZSA) is proposed. This algorithm computes delay bounds for each net based on a tentative set of connection delays chosen so that all timing requirements are met. ZSA chooses maximal delays bounds so that a delay increase on any net connection would produce a timing violation. Based on the delay upper bounds, the wirelength constraints can be generated. Net constraint generation is formulated as a linear programming (*LP*) problem which maximizes the range of permissible length for each net, subject to the *LP* constraints that timing requirements are met. Intuitively, ZSA will distribute extra slacks uniformly among connections on that path. After that, slacks are updated on other paths that are affected, and the process is repeated until every connection has zero slack. An improvement is suggested in [68], where a weighted slack budgeting is performed based on the delay per unit load function. A larger weight is assigned to nets which are more sensitive and the slack distribution is allocated proportionally to the weight.

Runtime improvement to slack budgeting using the non-zero slack allocation in intermediate steps is suggested in [38]. It omits recomputing slacks on connections whose slacks are altered by delay increase on the minimum-slack segment, and thus it converges faster than [42]. In practice, all slacks converge to near

zero in a few iterations. In [21], the Iterative-Minmax-PERT [68] procedure is generalized to guarantee the slacks go monotonically to zero.

In [55, 54] the delay budgeting problem is formulated as a convex programming problem with a special structure, thus efficient graph-based algorithm is proposed. It showed an average of 50% reduction in net length constraint violations over the well-known zero-slack algorithm [42]. In addition, different delay budgeting objective functions are studied and showed that performance improvements can be made without loss of solution quality. In a recent work [23], a new theoretical framework is presented which unifies several previous timing budget problems including: timing budgeting for maximizing total weighted delay relaxation, minimizing maximum relaxation, and min-skew time budget distribution. Dragon [67] uses design hierarchy information to compute *NLC*s and it is evaluated using an industrial place and route flow.

4.1.3 Incremental *NLC* Generation

Some *NLC* generation heuristics have taken an incremental approach to creating *NLC*s [10, 22]. These heuristics are used with incremental or iterative placement techniques. Initially, a loose set of *NLC* on a subset of nets is created, which may not yield a placement which meets timing requirements. Further iterations refine *NLC*s, tightening the bounds on nets critical at each iteration, so the slack is incrementally improved. Proponents of this approach argue that it is better than deriving a single shot *NLC* set. During an industry design flow, timing constraints are often unmeetable, even if every interconnect length is zero. Furthermore, a the set of *NLC*s that guarantee performance requirements may not be achievable be any placement.

An incremental transfer function which uses a linear programming based net constraint generation technique is proposed in [10]. The technique incrementally generates net constraints and iteratively reduces the length of critical nets by small increments. The goal of this LP-based technique is to derive a set of net constraints which will improve critical path delay $d_{initial}$ by a small amount, Δt. The k longest paths, p_i with delay $d_i > d_{goal}$ are selected, where $d_{goal} = d_{initial} - \Delta t$. For each path, p_i with delay d_i, the delay must be reduced by $d_i - d_{goal}$. Since the algorithm begins with an initial placement, the current horizontal and vertical lengths, Bx_i and By_i of bounding box wire length of each net n_i are known. In each iteration, the horizontal and vertical reduction goals, Δx_i, and Δy_i, are computed. The objective function is to minimize the total horizontal and vertical wirelength reduction.

$$min : \sum_{i \in Nets} (\Delta x_i + \Delta y_i). \tag{21}$$

For each path, a constraint is created in the linear program. For example, if path p_1 is composed of nets n_1, n_2, and n_3, the constraint would be:

$$(c_{1x} \cdot \Delta x_1 + c_{1y} \cdot \Delta y_1) + (c_{2x} \cdot \Delta x_2 + c_{2y} \cdot \Delta y_2) + (c_{3x} \cdot \Delta x_3 + c_{3y} \cdot \Delta y_3) < d_1 - d_{goal} \tag{22}$$

where c_{1x} and c_{1y} estimate the delay change per unit horizontal and vertical length of net n_1, and so on.

Additional constraints are imposed on each Δx_i and Δy_i reduction goal

$$\begin{aligned} \Delta x_i &< p \cdot Bx_i \\ \Delta y_i &< p \cdot By_i \end{aligned}$$

where p is a parameter ($0 < p < 1$), usually chosen to start with small value and increased if no solution is found to the LP. Since a net may be shared by more than one path, these constraints may limit the reduction goal of a shared net and force larger improvement goals in other nets.

A convex programming approach to net constraint generation is employed by [22]. Similar to the previous approach [10], it enumerates a set of critical paths to be considered and forms a set of linear constraints on the net delay of these paths. Unlike [10], each path must have an arrival time that is less than the required time. The result is a set of constraints that, if met, will result in zero slack for the paths considered.

4.2 Net Constraint Placement

Once net constraints are generated, placers must efficiently meet the constraints while generating legal placements and optimizing wire length. Net constraint placement algorithms have been proposed for many global and detail placement algorithms. This section explores two global placement approaches: partitioning and force directed, a several detailed placement approaches.

4.2.1 Partition-Based Net Constraint Placement

Several adaptations of the popular partitioning approach to global placement have been made for net constraint placement [62, 22, 63, 24]. This section examines a min-cut based approach [22] and two analytical partitioning based approaches [63, 24].

A modified min-cut partitioning based net constraint global placer is presented in [22]. The placer modifies the common min-cut partitioner using cut weights on constrained nets to change their cut cost. The weights are computed at each partitioning iteration based on the estimated net lengths. For each constrained net, the maximum and minimum estimated lengths max_i and min_i, are computed. max_i and min_i are the half perimeter of the smallest bounding box enclosing all the cells in n_i in their worst and best assignments to their partition choices. A net weight, w_i, is assigned based on a comparison of these estimates to the bound of the net, b_i. If $b_i < min_i$, then $w_i = maxcrit$ is assigned to the net since any increase in the net length is undesirable. If $b_i > max_i$, $w_i = 0$ since regardless of assignment choices, the net will not exceed its bound. For nets with $max_i \geq b_i \geq min_i$, the weight is computed as

$$\lfloor \frac{(max_i - b_i)}{(max_i - min_i)} \cdot maxcrit + 0.5 \rfloor \tag{23}$$

The Fiduccia-Mattheyses algorithm [20] is used to make the partition assignments. The algorithm does not guarantee that the net constraints will be met.

One of the first net constraint based global placers was published in [63]. Its general flow follows Proud [64], a partitioning placer that uses Mathematical Programming to determine partition assignments. Net constraints are created using the zero-slack algorithm [42] discussed in Section 4.1.2. To meet the *NLC*s, an iterative solving approach is used. At each iteration, a Lagrange multiplier is computed for each net. For each pin of a net, the multiplier is based on the length constraint, the nets current length, the previous pin weight and the sum of the weights of the other pins of its cell. This paper was the first to recognize that the other connectivity of a cell is important in computing pin weight.

While most net constraint partitioning placers model the *NLC*s directly in the partitioning assignment, a different approach is taken in [24]. This placer assumes that a preliminary wirelength driven partitioning assignment has been made already and it uses a linear programming formulation to make minimal reassignment to meet *NLC*s. Each net is modeled using a bounding box formulation. The location of each cell is restricted to lie within the boundaries of its parent partition and a "reassignment variable" is used to indicate if the cell is moved from its currently assigned partition or the other child partition of its parent. If the reassignment causes area violation, unconstrained cells are reassigned from the over capacity partition to the other child partition of its parent. The placer uses the analytical partitioning flow from Gordian [36].

4.2.2 Force Directed Net Constraint Placement

A force directed placer which optimizes for net constraints is presented in [48]. As with the other net constraint placers, this too builds on a strong wirelength driven placer, Kraftwerk [18]. Kraftwerk uses a quadratic programming model to generate cell locations. Net constraints are met by generating a higher net weight for nets which are not meeting their *NLC*s. The increased weights are allocated to the pins which determine the current boundary of the net. The outer pins, in both the X and Y dimension are given higher

weights to reduce its length as long as it does not meet its *NLC*. Another idea presented in this paper it to constrain the net segment connecting the nets driver to its critical receiver.

4.2.3 Net Constraint Based Detailed Placement

Several net constraint detailed placement algorithms have been proposed [26, 29, 10]. In [29], the ripple-move algorithm from Mongrel [30] is adapted to include the cost of nets which are violating their constraints. In [26], net constraint driven versions of Simulated Annealing [61, 58, 57, 60] and Domino [15] are proposed. The change to Simulated Annealing is a very simple addition to the SA cost function which reflects the cost of nets not meeting their *NLC*. The Domino transportation cost function is changed and several new techniques to recombine the fractured sub-cells are proposed.

A local movement approach which employs linear programming to reduce nets with constraints while minimizing the movement of unconstrained nets is presented in [10]. The objective function minimizes the squared movement of the center of a net's bounding box. This approach will create overlaps that must be resolved through a legalization phase which is not net constraint aware.

5 Path (or Timing Graph) Based Approach

Historically, path-based timing driven placement refers to those algorithms that directly model the timing constraints (which are inherently path-based) during placement. It ensures that all the paths under consideration will meet their timing requirements after placement. The benefit of path based approach is that it is *explicitly* timing-driven, unlike net based approaches which are *implicitly* timing-driven by converting timing constraints into net weights or wirelength constraints. The downside of this approach is the complexity of directly modeling timing in placement, as the number of paths may be prohibitive [7]. Except some early works such as simulated annealing [61], enumerating all paths are not widely adopted. To make the problem size small, one can select only the near-critical paths, but even that could still be huge. The potential problem of only selecting a set of critical paths is that some non-critical paths may become critical.

A more powerful technique is to embed *timing graph* (through a built-in simplified version of static timing analyzer) into the timing-driven placement formulation. It implicitly considers *all* topological paths and formulates them into some mathematical programming framework by introducing intermediate auxiliary variables (such as arrival times). It eliminates the need to enumerate/optimize a limited set of paths. The linear programming (LP) based formulation is popular as the half parameter wirelength model can be formulated exactly into an LP framework. To explicitly write down the delay modeling and timing propagation with respect to the cell locations (x, y), simple/linearized models are often used. In this section, we will first review the general LP-based formulation (which can easily be extended to handle non-linear mathematical programming). Then we discuss various techniques such as partitioning-based overlap removal and Lagrangian relaxation to complement the general LP-based formulation. We also discuss the Simulated Annealing technique for path-based timing driven placement and a recent technique using differential timing analysis.

5.1 The LP-Based Formulation

The general linear programming (LP) based formulation consists of two sets of variables and constraints - physical and electrical. The physical variables/constraints deal with variables and equations representing cell locations and net lengths (e.g., computed through the half-perimeter wire length model). The electrical variables/constraints deal with gate and net delay models, arrival time propagation through the critical path method, and constraints that all required arrival times at timing endpoints are met. The objective function may be maximizing either worse or total negative slack, or weighted wirelength, etc.

5.1.1 Physical Constraints

For cell i, its center coordinates x_i, y_i are the variables of the LP program. For a net e_j, let l_j, r_j, t_j and b_j represent its left, right, top, and bottom locations of its bounding box. Let N_j denote the set of cells connected to net e_j, then we have

$$\begin{aligned} l_j &\leq x_i + pin_x(i,j) \\ r_j &\geq x_i + pin_x(i,j) \\ t_j &\leq y_i + pin_y(i,j) \\ b_j &\geq y_i + pin_y(i,j), \quad \forall i \in N_j \end{aligned} \tag{24}$$

where $pin_x(i,j)$ and $pin_y(i,j)$ are the pin offsets of cell i for its pin connecting to net e_j in horizontal and vertical directions, respectively. The HPWL of net e_j is represented by L_j

$$L_j = r_j - l_j + t_j - b_j \tag{25}$$

5.1.2 Electrical/Timing Constraints

Let the gate delay $GDelay_i(k,o)$ represent the pin delay from an input pin k to output pin o of cell i. It can be modeled as a linear function of the load capacitance at the output pin and the slope (transition time) at the input pin with a reasonably high degree of accuracy. Similarly, the slope at the output pin of cell i can be described by a linear function.

$$\begin{aligned} GDelay_i(k,o) &= a_0 + a_1 \cdot CLoad_i(o) + a_2 \cdot Slope_i(k) \\ Slope_i(o) &= b_0 + b_1 \cdot CLoad_i(o) + b_2 \cdot Slope_i(k) \end{aligned}$$

where $Slope_i(k)$ is the slope at the input pin k of cell i, and $Slope_i(o)$ is the slope at the output pin o of cell i, and $CLoad_i(o)$ is the capacitance load seen by the output pin o. The constants a_0, a_1, a_2, b_0, b_1, and b_2 are determined by standard cell library characterizations. These delay and output slope equations can be defined for every feasible signal transition for the cell.

The delay for net e_j, $NDelay_j(i_1,o,i_2,k)$ from output pin o of cell i_1 to the input pin k of cell i_2 is modeled in the LP using a simplified Elmore model [19] by the following equation:

$$NDelay_j(i_1,o,i_2,k) = K_D \cdot r \cdot L_j \cdot (\frac{c \cdot L_j}{2} + CLoad_{i_2}(k)) \tag{26}$$

where r is the unit resistance of the interconnect, c is the unit capacitance constant, and K_D is a constant, 0.69 [11]. If the resistance and capacitance in the horizontal and vertical directions are not equal, an alternate model can be used which replaces L_j with individual variables for the horizontal and vertical lengths.

The arrival time at each pin is modeled through timing propagation and critical path method. Two types of equations are used, the first for input pins and the second for output pins. For input pin k of cell i_2, its arrival time is

$$Arr_{i2}(k) = Arr_{i1}(o) + NDelay_j(i_1,o,i_2,k) \tag{27}$$

The arrive time at an output pin o of cell i is represented by the *LP* variable $Arr_i(i,o)$ and a set constraints, one for each input pin of cell i. Assuming two input pins k_1 and k_2 for cell i, the equations would be:

$$Arr_i(k_1) + GDelay_i(k_1,o) \leq Arr_i(o) \tag{28}$$

$$Arr_i(k_2) + GDelay_i(k_2,o) \leq Arr_i(o) \tag{29}$$

Most implementations assume the arrival time at the output of a sequential cell to be zero.

Each library cell has a maximum drive strength, limiting the total capacitance the cell can drive. This drive strength limit is incorporated in the *LP* through length limits on the driven net. This limit is a pre-computed constant to the *LP* formulation.

$$L_j < CMax(e_j) \tag{30}$$

5.1.3 Objective Functions

The required time at input pin k of sequential cell v_i, $Req_i(k)$, is a constant input. The negative slack at these timing endpoints is represented by variable $Slk_i(k)$ and equations

$$Slk_i(k) <= Req_i(k) - Arr_i(k) \tag{31}$$

$$Slk_i(k) \leq 0 \tag{32}$$

The second constraint is needed so that paths are not optimized beyond what is required to meet timing. This constraint can be adapted so that a slight positive margin is created for each path.

The path-based timing driven placement can optimize the total negative slack, i.e.,

$$max : \sum_{i \in sequential} Slk_i(k) \tag{33}$$

To optimize the worst negative slack, a variable representing the worst negative slack is introduced, WNS, i.e.,

$$WNS < Slk_i(k) \tag{34}$$

And the objective function is simply:

$$max : WNS \tag{35}$$

The LP-based objective function can also be a combination of wirelength and slack [31], e.g.,

$$min : \sum L_j - \alpha \cdot WNS \tag{36}$$

where α is the weight to tradeoff wirelength and worst negative slack.

To summarize, the complete LP formulation for timing driven placement can be written in the following generic term

$$\begin{aligned} Minimize \quad & f(\mathbf{X}) \\ subject\ to \quad & A\mathbf{X} \leq D \end{aligned} \tag{37}$$

where $\mathbf{X}$ is the set of variables including gate coordinates and auxiliary variables, $f(\mathbf{X})$ is the objective function which can be (33), (35), or (36). $A\mathbf{X} \leq D$ includes all the physical and electrical constraints such as net bounding box constraints, delay constraints, slack constraints, and other possible additional constraints (such as the center of gravity constraints as in [31]).

5.2 Partitioning Based Overlap Removal

The LP-based formulation may create a lot of overlaps. Partitioning based approach can be used together with LP-based formulation to remove the cell overlaps, as proposed in the original timing graph based placer Allegro [31]. At each partitioning step, it formulates a *LP* problem to determine locations of cells. Each partition is divided into two sub-partitions, and its cells are sorted based on the *LP* locations to determine

the new partition assignment. The *LP* model is similar to Section 5.1. The objective function is similar to (36). The factor α is used to trade off timing optimization versus wirelength. Additional physical constraints includes center of gravity constraint and partition boundary constraint. The center of gravity constraint, as shown in (38), tries to place the center of gravity of all the gates in the same partition to be in the center of the partition, while the boundary constraints prevent gates being placed outside the partition boundaries.

$$\overline{x} = \frac{\sum m_i x_i}{m_i} \tag{38}$$

where $\overline{x}$ represents the center of the partition in x direction; x_i is the position of gate i; and m_i is the equivalent mass of gate i, approximated by the gate width.

5.3 Lagrangian Relaxation Method

The number of constraints in the general LP-based formulation in (37) can be enormous, even for moderate size circuits. Lagrangian relaxation is a very effective technique to transform the original constrained LP-formulation into a set of unconstrained problems in an iterative manner, e.g., as in [59]. While the objective function used in [59] is the quadratic wirelength, the principle of Lagrangian relaxation method is the same. For the general mathematical programming formulation in (37), suppose A has m constraint equations. We can define a size-m vector Lagrange multipliers λ and add the non-negative term $\lambda \cdot (D - AX)$ to the objective function:

$$max_{\lambda} min_{X} f(\mathbf{X}) + \lambda \cdot (D - A\mathbf{X}) \tag{39}$$

When λ is fixed, minimizing $f(\mathbf{X}) + \lambda(D - A\mathbf{X})$ is an unconstrained mathematical programming problem, which can be solved efficiently. Then the Lagrange multiplier λ will be updated to solve a new unconstrained optimization problem. This process is iterated to obtain the constrained optimal solution.

5.4 Simulated Annealing

The simulated annealing is a generic probabilistic algorithm for global optimization. It randomly moves gates, and accepts or reject the move based on certain cost function. It is very flexible, i.e., it can take any objective function and consider accurate timing models, if needed. In [61], the Simulated Annealing algorithm is used for timing driven placement by augmenting the cost function to include path based timing information. Since efficient runtime of the cost evaluation step is critical in *SA*, great care has to be taken in implementing the timing cost function. Rather than updating the static timing graph whenever a cell is moved, the approach in [61] uses an enumerated set of critical paths, $P_{critical}$. During a move cost evaluation, the paths impacted can be directly updated by adding the change in delay for the nets connected moved cells. The SA engine has two loops. The outer loop identifies $P_{critical}$, and the inner loop runs a number of annealing iterations. In each outer loop of the annealing process, $P_{critical}$ is chosen as the K most critical paths using Dreyfus method [16]. In the inner loop, the nets impacted by a move will update the slack of paths, and the total timing cost is the sum of the path slacks in $P_{critical}$. When the inner loop finishes, the outer loop updates the critical paths with new gate locations, and continue the inner loop. The simulated annealing cost function is a combination of wirelength cost and timing cost function.

5.5 Graph Based Differential Timing

A recent work by Chowdary et al. [11] addresses the correlation problem of graph based placers with final sign-off timers. Rather than modeling and computing delays and arrival times as was presented above, this approach optimizes an initial global placement based on the differences in delays, arrival and required times at all pins of a circuit, relative to a reference static timing analysis. It terms this approach *differential timing analysis* [11]. This differential timing analyzer is almost exact in the neighborhood of the reference static

timing, including modeling of setup time and latch transparency. It also introduces another improvement to Graph timing based placement. The constants used in the delay and slope equations (26) are only accurate for a range of values of output loads and input slopes. To maintain the validity of the differential timing model, placement changes are limited to a local neighborhood. It then solves several iterations of the *LP* adjusting model constants and the neighborhood limits in each iteration. Differential timing is optimized using *LP*. A set of *LP* equations which parallel the static timing graph equations are used. For example, the delta wirelength can be obtained by

$$\Delta L_j = r_j - l_j + t_j - b_j - L_j^{old} \tag{40}$$

where L_j^{old} is the wirelength of net j in the current placement. The equations for $\Delta delay$, $\Delta slope$, $\Delta arrival$, and $\Delta slack$ can be formed similarly [11].

6 Additional Techniques

There are many additional timing-driven placement algorithms in the literature which do not fall exactly into the previous classifications. As mentioned earlier, net-based and path-based algorithms all have pros and cons. A hybrid approach is proposed recently [39] to combine the net weighting and net constraints together with LP-based formulations. Furthermore, due to the complexity of modern placement problems and the iterative refinement nature from global placement to detailed/legal placement, it is very important to have stability between placement iterations. In this section, we present several representative and recent techniques for timing-driven and timing-aware placement.

6.1 Hybrid Net and Path Based Approach

In [39], a hybrid approach is proposed to combine the net weighting and net constraints together with LP-based formulations. The net-based approaches, especially the net weighting, have low computational complexity and high flexibility/scalability. Therefore, net-based approaches have more advantages as the circuit complexity continues to increase. However, net weighting often completely ignores slew propagation. Since timing is inherently path based, an effective net weighting algorithm should be based on path analysis and consider timing propagation. Furthermore, net-based approaches are often done in an ad-hoc manner and may have problems with convergence. For instance, while the delay on critical paths decrease, other paths become critical, and this leads to a convergence problem. A systematic way of explicit perturbation control is important for net-weighting based algorithms. The hybrid approach in [39] uses a hybrid net and path-based delay sensitivity with limited-stage slew propagation as basis for net weighting. The objective function is the weighted wirelength for a set of critical paths. The LP formulation considers not only cells on the timing-critical paths, but also cells that are logically adjacent to the critical paths in a unified manner, through weighted LP objective function and net bound constraints. This approach is suitable for incremental timing improvement.

6.2 Hippocrates: A Detailed Placer Without Degrading Timing

Another timing driven incremental placement algorithm [49] helps to reduce total wirelength and improve timing at the same time. It specifically maintains the timing constraints while reducing wirelength during detailed placement. The detailed placement algorithms it uses can be any commonly used move-based transforms, i.e., cell swapping, cell moving, etc. Instead of modeling path constraints, it models the timing constraints at each input pin. The advantage of this is that it reduces the computation complexity, which allows it to model timing constraints on every timing path. Therefore the output of this algorithm guarantees no timing degradation. The timing constraint on each pin is called *delta arrival time* constraint, which is

defined as the difference of arrival time at this pin to the arrival time of the most critical input pin on this gate. By constraining the delta delay changed by moving cells to be less than the delta arrival time on each pin, it guarantees that the final arrival time at timing end points would not degrade. It also models slew and load capacitance constraints. Experimental results [49] show that Hippocrates helps improve wirelength and timing significantly, in particular on total negative slack, while conventional detailed placement algorithms fail to maintain the original timing.

6.3 Accurate Net Modeling Issue

While most timing driven placers assume simple net models, some uses specialized net models for timing critical nets, e.g., during global placement [44] or detailed placement [1]. The first, [44], based on force-directed global placement [18], proposes a more accurate tree net model to replace the ubiquitous clique/star net models normally used in quadratic placers. A Steiner tree net model is constructed and the length of each tree segment is controlled by weighting the individual segments to improve timing. This new model does not increase numerical complexity. This net model is not specific to the force-directed formulation and could be used in other quadratic programming (QP) based placers. In order to determine the weight of each Steiner segment, the segment sensitivity is computed by determining the net delay derivative with respect to the segment length. In this way the segments which produce the most slack improvement are shortened the most.

Another work [1] proposes simultaneous detailed placement and routing to optimize timing. The algorithm is stable and incremental, and it reduces WNS by 9-14%, although the runtime is quite high. It begins with a placed and global routed netlist and optimizes the k most critical paths using a non-convex mathematical programming model that optimizes slack while capturing the timing impact of cell movements and Steiner point changes of the global route. In this approach, cell movements may change the Steiner tree topology. Within the solving steps, each net is analyzed to ensure that its Steiner tree is correct, otherwise a new topology is generated. Since routing changes are modeled, this is a more accurate net model than those commonly used net models discussed in previous sections.

7 Conclusions

While timing-driven placement has been studied extensively in the past two decades, the problem is still far away from being solved [14]. Many challenges still remain due to the ever-growing problem size and complexity. On one hand, modern system-on-chip designs have millions of placeable cells and hundreds/thousands of macros [43]; on the other hand, stringent timing requirements and physical effects pose increasing challenges to the timing closure where timing driven placement plays a key role.

It shall be noted that to achieve the overall timing closure, timing driven placement needs to work closely with synthesis/optimization tools (such as buffer insertion and gate sizing) and routing (in particular global routing). The entire physical design/synthesis closure is an extremely complex task. Furthermore, modern complex SOC designs usually have multiple clock domains, or even multiple cycle paths which make the timing-driven placement problem even more complicated. Due to the infrastructure limitation, the academia has not been able to fully push the state-of-the-art and limits of timing-driven placement. With the availability of OpenAccess [27] and the OpenAccess Gear timer [65, 66], it is possible to push the frontier of the very successful ISPD Placement Contest [43] for university researchers to work on more realistic timing objectives. As technology scales into sub-100nm regimes, new physical and manufacturing effects, in particular leakage/power and variations have to be considered together with timing closure during timing-driven placement [9, 35], which requires continuous innovations for better quality and productivity.

References

[1] A. H. Ajami and M. Pedram. Post-layout timing-driven cell placement using an accurate net length model with movable steiner points. In *Proc. Asia and South Pacific Design Automation Conf.*, pages 595–600, 2001.

[2] A. Marquardt V. Betz and J. Rose. Timing driven placement for FPGA. In *ACM Symposium on FPGA*, pages 203–213, 2000.

[3] Kenneth D. Boese, Andrew B. Kahng, and Stafanus Mantik. On the relevance of wire load models. In *Proceedings of the 2001 international workshop on System-level interconnect prediction*, pages 91–98. ACM Press, 2001.

[4] M.A. Breuer, M. Sarrafzadeh, and F. Somenzi. Fundamental CAD algorithms. *IEEE Transactions on Computer-Aided Design*, 19(12):1449–1475, 2000.

[5] M. Burstein and M. N. Youssef. Timing influenced layout design. In *Proc. Design Automation Conf.*, pages 124–130, 1984.

[6] Tony Chan, Jason Cong, and Kenton Sze. Multilevel generalized force-directed method for circuit placement. In *Proc. Int. Symp. on Physical Design*, pages 185–192, New York, NY, USA, 2005. ACM Press.

[7] C.C. Chang, J. Lee, M. Stabenfeldt, and R.S. Tsay. A practical all-path timing-driven place and route design system. In *Asia-Pacific Conference on Circuits and Systems*, pages 560–563. IEEE/ACM, 1994.

[8] 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 *Proc. Int. Symp. on Physical Design*, pages 155–160, April 2002.

[9] Yongseok Cheon, Pei-Hsin Ho, Andrew B. Kahng, Sherief Reda, and Qinke Wang. Power-aware placement. In *Proc. Design Automation Conf.*, pages 795–800, New York, NY, USA, 2005. ACM Press.

[10] W. Choi and K. Bazargan. Incremental placement for timing optimization. In *Proc. Int. Conf. on Computer Aided Design*, pages 463–466, 2003.

[11] 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. Design Automation Conf.*, pages 801–806, 2005.

[12] J. Cong, J. R. Shinnerl, M. Xie, T. Kong, and X. Yuan. Large-scale circuit placement. In *ACM Transactions on Design Automation of Electronic Systems*, pages 389–430, 2005.

[13] Jason Cong, Lei He, Cheng-Kok Koh, and Patrick H. Madden. Performance optimization of VLSI interconnect layout. *Integration, the VLSI Journal*, 21:1–94, 1996.

[14] Jason Cong, Michail Romesis, and Min Xie. Optimality and stability study of timing-driven placement algorithms. In *Proc. Int. Conf. on Computer Aided Design*, page 472, Washington, DC, USA, 2003. IEEE Computer Society.

[15] K. Doll, F. M. Johannes, and K. J. Antreich. Iterative placement improvement by network flow methods. *IEEE Transactions on Computer-Aided Design*, 13:1190–1200, 1994.

[16] S.E. Dreyfus. An appraisal of some shortest-path algorithms. *Operations Research*, 17:395– 412, 1969.

[17] A. E. Dunlop, V. D. Agrawal, D. N. Deutsch, M. F. Jukl, P. Kozak, and M. Wiesel. Chip layout optimization using critical path weighting. In *Proc. Design Automation Conf.*, pages 133–136, 1985.

[18] H. Eisenmann and F. M. Johannes. Generic global placement and floorplanning. In *Proc. Design Automation Conf.*, pages 269–274, 1998.

[19] W. C. Elmore. The transient response of damped linear networks with particular regard to wide-band amplifiers. *Journal of Applied Physics*, 19(1):55–63, January 1948.

[20] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the Design Automation Conference*, pages 175–181, 1982.

[21] J. Frankle. Iterative and adaptive slack allocation for performance-driven layout and FPGA routing. In *Proc. Design Automation Conf.*, pages 539–532, 1992.

[22] T. Gao, P. M. Vaidya, and C. L. Liu. A performance driven macro-cell placement algorithm. In *Proc. Design Automation Conf.*, pages 147–152, 1992.

[23] S. Ghiasi, E. Bozorgzadeh, S. Choudhuri, and M. Sarrafzadeh. A unified theory of timing budget management. In *Proc. Int. Conf. on Computer Aided Design*, pages 653–659, 2004.

[24] B. Halpin, C. R. Chen, and N. Sehgal. Timing driven placement using physical net constraints. In *Proc. Design Automation Conf.*, pages 780–783, 2001.

[25] B. Halpin, C. Y. R. Chen, and N. Sehgal. A sensitivity based placer for standard cells. In *Proc. the 10th Great Lakes Symp. on VLSI*, pages 193–196, 2000.

[26] B. Halpin, C. Y. R. Chen, and N. Sehgal. Detailed placement with net length constraints. In *Proc. the 3rd International Workshop System on Chip*, page 22, 2003.

[27] http://openeda.si2.org/.

[28] D. J. H. Huang and A. B. Kahng. Partition-based standard-cell global placement with an exact objective. In *Proc. Int. Symp. on Physical Design*, pages 18–25, 1997.

[29] S. Hur, T. Cao, K. Rajagopal, Y. Parasuram, A. Chowdhary, V. Tiourin, and B. Halpin. Force directed mongrel with physical net constraints. In *Proc. Design Automation Conf.*, pages 214–219, 2003.

[30] S. Hur and J. Lillis. Mongrel: Hybrid techniques for standard cell placement. In *Proceedings of the International Conference on Computer-Aided Design*, pages 165–170. IEEE, 2000.

[31] M. A. B. Jackson and E. S. Kuh. Performance-driven placement of cell based ic's. In *Proc. Design Automation Conf.*, pages 370–375, 1989.

[32] A. B. Kahng, S. Mantik, and I. L. Markov. Min-max placement for large-scale timing optimization. In *Proc. Int. Symp. on Physical Design*, pages 143–148, 2002.

[33] A. B. Kahng, I. L. Markov, and S. Reda. Boosting: min-cut placement with improved signal delay. In *Proc. Design, Automation and Test in Eurpoe*, pages 1098–1103, 2004.

[34] A. B. Kahng and Q. Wang. Implementation and extensibility of an analytic placer. *Proc. Int. Symp. on Physical Design*, pages 18–25, April 2004.

[35] Andrew B. Kahng, Chul-Hong Park, Puneet Sharma, and Qinke Wang. Lens aberration aware timing-driven placement. In *Proc. Design, Automation and Test in Eurpoe*, pages 890–895, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association.

[36] J. M. Kleinhans, G. Sigl, F. M. Johannes, and K. J. Antreich. Gordian: VLSI placement by quadratic programming and slicing optimization. *IEEE Transactions on Computer-Aided Design*, 10(3):356–365, 1991.

[37] T. Kong. A novel net weighting algorithm for timing-driven placement. In *Proc. Int. Conf. on Computer Aided Design*, pages 172–176, 2002.

[38] W. K. Luk. A fast physical constraint generator for timing driven placement. In *Proc. Design Automation Conf.*, pages 626–631, 1991.

[39] T. Luo, D. Newmark, and D. Z. Pan. A new LP based incremental timing driven placement for high performance designs. In *Proc. Design Automation Conf.*, 2006.

[40] Naresh Maheshwari and Sachin Sapatnekar. *Timing Analysis and Optimization of Sequential Circuits*. Kluwer Academic Publishers, 1999.

[41] M. Marek-Sadowska and S. P. Lin. Timing driven placement. In *Proc. Int. Conf. on Computer Aided Design*, pages 94–97, 1989.

[42] R. Nair, L. Berman, P. S. Hauge, and E. J. Yoffa. Generation of performance constraints for layout. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 8(8):860–874, 1989. (ICCAD'87).

[43] Gi-Joon Nam. ISPD 2006 placement contest: Benchmark suite and results. In *Proc. Int. Symp. on Physical Design*, pages 167–167, New York, NY, USA, 2006. ACM Press.

[44] B. Obermeier and F. M. Johannes. Quadratic placement using an improved timing model. In *Proc. Design Automation Conf.*, pages 705–710, 2004.

[45] S. Ou and M. Pedram. Timing-driven bipartitioning with replication using iterative quadratic programming. In *Proc. Asia and South Pacific Design Automation Conf.*, pages 105–108, 1999.

[46] S. Ou and M. Pedram. Timing-driven placement based on partitioning with dynamic cut-net control. In *Proc. Design Automation Conf.*, pages 472–476, 2000.

[47] Lawrence T Pillage and Ronald A Rohrer. Asymptotic waveform evaluation for timing analysis. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 9:352–366, April 1990.

[48] K. Rajagopal, T. Shaked, Y. Parasuram, T. Cao, A. Chowdhary, and B. Halpin. Timing driven force directed placement with physical net constraints. In *Proc. Int. Symp. on Physical Design*, pages 60–66, 2003.

[49] H. Ren, D. Z. Pan, C. Alpert, G.-J. Nam, and P. Villarrubia. Hippocrates: First-Do-No-Harm detailed placement. In *Proc. Asia and South Pacific Design Automation Conf.*, January 2007.

[50] H. Ren, D. Z. Pan, and D. Kung. Sensitivity guided net weighting for placement driven synthesis. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, pages 711–721, May 2005. (ISPD'04).

[51] B. M. Riess and G. G. Ettelt. SPEED: fast and efficient timing driven placement. In *Proc. IEEE Int. Symp. on Circuits and Systems*, pages 377–380, 1995.

[52] Sr. Robert B. Hitchcock. Timing verification and the timing analysis program. In *Proceedings of the Design Automation Conference*, pages 594–604, 1982.

[53] Jarrod A. Roy, James F. Lu, and Igor L. Markov. Seeing the forest and the trees: Steiner wirelength optimization in placemen. In *ISPD '06: Proceedings of the 2006 international symposium on Physical design*, pages 78–85, New York, NY, USA, 2006. ACM Press.

[54] M. Sarrafzadeh, D. Knol, and G. Tellez. A delay budgeting algorithm ensuring maximum flexibility in-placement. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 16(11):1332–1341, November 1997.

[55] M. Sarrafzadeh, D. Knol, and G. Tellez. Unification of budgeting and placement. In *Proc. Design Automation Conf.*, pages 758–761, 1997.

[56] Prashant Saxena and Satyanarayan Gupta. Shield count minimization in congested regions. In *Proceedings of 2002 International Symposium on Physical Design*, pages 78–83. ACM Press, 2002.

[57] C. Sechen and A.S. Vincentelli. The Timberwolf placement and routing package. In *IEEE Custom Integrated Circuits Conference*, pages 522–527, 1984.

[58] Carl Sechen. *VLSI Placement and Global Routing Using Simulated Annealing*. Kluwer, B.V., 1988.

[59] A. Srinivasan, K. Chaudhary, and E. S. Kuh. Ritual: A performance driven placement algorithm for small cell ics. In *Proc. Int. Conf. on Computer Aided Design*, pages 48–51, 1991.

[60] W. J. Sun and C. Sechen. A loosely coupled parallel algorithm for standard cell placement. In *Proceedings of the International Conference on Computer-Aided Design*, pages 137–144. IEEE, 1994.

[61] W. Swartz and C. Sechen. Timing driven placement for large standard cell circuits. In *Proc. Design Automation Conf.*, pages 211–215, 1995.

[62] Masayuki Terai, Kazuhiro Takahashi, and Koji Sato. A new min-cut placement algorithm for timing assurance layout design meeting net length constraint. In *DAC '90: Proceedings of the 27th ACM/IEEE conference on Design automation*, pages 96–102, New York, NY, USA, 1990. ACM Press.

[63] R. S. Tsay and J. Koehl. An analytic net weighting approach for performance optimization in circuit placement. In *Proc. Design Automation Conf.*, pages 636–639, 1991.

[64] Ren-Song Tsay, Ernest S. Kuh, and Chi-Ping Hsu. Proud: A fast sea-of-gates placement algorithm. In *Proceedings of the Design Automation Conference*, pages 318–323. IEEE Computer Society Press, 1988.

[65] Z. Xiu and R. A. Rutenbar. Timing-driven placement by grid-warping. In *Proc. Design Automation Conf.*, pages 585–590, 2005.

[66] Zhong Xiu, David A. Papa, Philip Chong, Christoph Albrecht, Andreas Kuehlmann, Rob A. Rutenbar, and Igor L. Markov. Early research experience with openaccess gear: an open source development environment for physical design. In *Proc. Int. Symp. on Physical Design*, pages 94–100, New York, NY, USA, 2005. ACM Press.

[67] X. Yang, B. Choi, and M. Sarrafzadeh. Timing-driven placement using design hierarchy guided constraint generation. In *Proc. Int. Conf. on Computer Aided Design*, pages 177–180, 2002.

[68] H. Youssef and E. Shragowitz. Timing constraints for correct peformance. In *Proc. Int. Conf. on Computer Aided Design*, pages 24–27, 1990.