Local Search Algorithms for Timing-Driven Placement under Arbitrary Delay Models

Adrian Bock, Stephan Held, Nicolas Kämmerling, and Ulrike Schorr
Research Institute for Discrete Mathematics
Lennéstr. 2
53113 Bonn, Germany
{held,kaemmerling,schorr}@or.uni-bonn.de

April 10, 2015

We present local search algorithms for timing-driven placement optimization. They find local slack optima for cells under arbitrary delay models and can be applied late in the design flow.

The key ingredients are an implicit path straightening and a clustering of neighboring cells. Cell clusters are moved jointly to speed up the algorithm and escape suboptimal solutions, in which single cell algorithms are trapped, particularly in the presence of layer assignments. Given a cell cluster, we initially perform a line search for maximum slack on the straight line segment connecting the most critical upstream and downstream cells of the cluster. Thereby, the Euclidean path length is minimized. An iterative application will implicitly straighten the path. Later, slacks are improved further by applying ascent steps in estimated supergradient direction.

The benefit of our algorithms is demonstrated experimentally within an industrial microprocessor design flow, and on recent ICCAD benchmarks circuits.

Keywords: timing-driven placement, local search, accurate delay models

1 Introduction

Timing-driven placement is a central problem for achieving timing closure in VLSI design. The traditional approach to timing-driven placement is to assign weights or delay bounds to critical nets or individual pin-to-pin connections and employ a placement algorithm minimizing the weighted wirelength [4, 6, 7, 13, 18, 20, 22, 23]. Internally, delay bounds are often realized by net weights, e.g. [22]. Alternatively, path based approaches model static timing constraints explicitly, approximating delays by linear or quadratic functions [5, 11, 24].

Most placement algorithms estimate net lengths by the half perimeter bounding box of their pins, which does not allow Steiner topology aware delay models. In [1, 16] Steiner points are placed by the placement algorithm allowing more accurate delay models. Recently, [8] introduced the simplifying γ-*net model* during global placement and linearized delays in a minimum-cost-flow formulation during legalization.

All these approaches help to improve critical path delays, but they require many iterations to obtain good net weights, bounds, or Steiner topologies. After some iterations, it can be more effective to move cells to a local optimum under an accurate delay model (ignoring overlaps), and then legalize the design.

For the local search with a single cell, linear programming has been proposed in [17], solvable by geometric algorithms [14]. Industrial design environments use single cell movements for improving timing under accurate delay models together with gate sizing and buffering [21].

We propose a new local search algorithm that locally maximizes slacks through a series of local moves under arbitrary delay models, e.g., linear, Elmore [9], or asymptotic waveform emulation (AWE) [19]. In general, delay functions are discontinuous, as delays depend on the Steiner tree topologies, which change with the placement. Hence, finding a placement that maximizes the worst slack is a non-concave optimization problem. However, local search can be a powerful tool to obtain good results.

The core algorithm moves single cells to a local optimum. However, this may stall in suboptimal solutions, particularly in the presence of layer assignments, as the examples in Figure 1 show. In Figure 1a, a critical path with a slack of -100 *ps* passes an inverter through a wide wire and then enters a register 'R' through a thin wire. For better timing, the inverter tends to the right, spanning more length by the low resistance interconnect

(a) register endpoint

(b) detour through multiple cells

Figure 1: Two situations which single cell moves cannot improve. Critical paths are drawn in red.

as indicated by the blue arrow. The register has a positive output slack of $100ps$ and tends to the left. However, both cells are mutually blocking a further movement. In Figure 1b, the critical (red) path contains a sequence of two high resistance nets in the middle. The entering net on the left and the leaving net on the right are assigned to low-resistance layers. Again the two inverters tend to the center to span more length with the low resistance nets, where the NAND blocks any progress. In both cases, clustering the cells in the green circle leads to an improving direction as indicated by the green arrows. As we will see in the experimental results, clusters also help to move cells jointly across placement blockages, where single cell moves fail.

In contrast to most existing approaches, we are not considering placement constraints explicitly, but just by employing standard legalization techniques. The main contributions of our local search algorithms are as follows:

- Slacks are addressed directly under arbitrary delay models.
- We present a new implicit path straightening algorithm,
- an estimated supergradient ascent to locally optimal cell locations, and
- a clustering of cells for escaping from single cell optima and speeding up the computations.

In Section 2, we describe the problem and define the central objective of the local search. The local search algorithms are presented in Section 3, followed by experimental results in Section 4.

2 Problem Formulation

Let $\mathcal{C}$ be the set of cells and P the set of pins in the design. Each cell $c \in \mathcal{C}$ is assigned a cell location $x_c = (x_c^h, x_c^v) \in \mathbb{R}^2$ and the vector $x = (x^h, x^v) \in (\mathbb{R}^2)_{c\in\mathcal{C}}$ describes all cell locations. We do not make any assumption on the underlying delay model and wire topology. Instead, we use a timing engine that, for a given vector x of locations returns the worst (late) slack at a pin $p \in P$ which is defined as the minimum difference of the delay bound and actual delay of a path through p.

In order to improve the delay of a critical path, we can reduce its length or balance load capacities w.r.t. drive strengths by moving cells on the path. In addition, the load capacitance of the nets on the critical path can be reduced by moving less critical downstream cells closer to the path. To accomplish this uniformly for all gates, regardless if they lie on a critical path or not, we use the local slack metric from [10]:

Given a cell $c \in \mathcal{C}$ and the current placement x, let $\mathrm{s}^+(c, x)$ be the worst slack at an output pin of c and $\mathrm{s}^-(c, x)$ be the worst slack of an upstream driver pin to a data input of c, ignoring slacks on clock inputs of registers. Obviously, $\mathrm{s}^+(c, x) \geq \mathrm{s}^-(c, x)$ for all cells but registers, since a path that determines $\mathrm{s}^+(c, x)$ must pass through one of the upstream driver pins.

Given a placement x, we define the *local timing-driven placement problem* for a cell $c \in \mathcal{C}$ as finding a new placement x' with $x_{c'} = x'_{c'}$ for all $c' \in \mathcal{C} - c$, maximizing its *local slack*

$$\mathrm{s}(c, x') := \min\{\mathrm{s}^-(c, x'), \mathrm{s}^+(c, x'), 0\}, \tag{1}$$

where we are assuming a slack threshold zero.

To tackle the *timing-driven placement problem*, which is to find a placement x maximizing $\min_{c\in\mathcal{C}} \mathrm{s}(c,x)$, we iteratively apply its local variant for critical cells.

3 Local Search Algorithms

At the core of our algorithm, we consider a single cell $c \in \mathcal{C}$ and move it to a cell location that maximizes its local slack $\mathrm{s}(c,x)$. There are two approaches to improve the slack. The first is to iteratively determine an ascent direction by an approximated supergradient. The second algorithm has the goal to implicitly straighten the critical path.

3.1 Supergradient Ascent

In this variant, we pretend that $\mathrm{s}(c,x)$ is a concave function of x_c, and compute a discretized supergradient $d_c(x)$, which we use as an ascent direction.

Let $c \in \mathcal{C}$ be the current cell and let $e^h, e^v \in (\mathbb{R}^2)^{\mathcal{C}}$ with $e^h_c = (1,0) \in \mathbb{R}^2$ and $e^v_c = (0,1) \in \mathbb{R}^2$ and zero everywhere else. For $\epsilon > 0$ let

$$\begin{aligned} x^l &= x - \epsilon \cdot e^h, \quad x^r = x + \epsilon \cdot e^h \\ x^d &= x - \epsilon \cdot e^v, \quad x^u = x + \epsilon \cdot e^v \end{aligned}$$

arise from x by moving cell c by ϵ to the left, right, down, and up. We compute a discretized ascent direction

$$\tilde{d}_c(x) := \frac{1}{2\epsilon} \begin{pmatrix} \mathrm{s}(c,x^r) - \mathrm{s}(c,x^l) \\ \mathrm{s}(c,x^u) - \mathrm{s}(c,x^d) \end{pmatrix} \in \mathbb{R}^2$$

and normalize it:

$$d_c(x) := \tilde{d}_c(x) \Big/ \left\| \tilde{d}_c(x) \right\|_1 \in \mathbb{R}^2. \tag{2}$$

We employ the normalized vector, because $\left\| \tilde{d}_c(x) \right\|$ is unpredictably large when $\mathrm{s}(c,x)$ is non-differentiable or even discontinuous at x. In the latter case the supergradient is not defined, but the discrete approximation can still yield an ascent direction.

The supergradient ascent method is summarized in Algorithm 1. To speed up the search, in particular for bridging long distances, we scale the step size dynamically. The step size is doubled whenever an improvement is found and it is halved if no improvement is found at the current step size.

The If-statement in line 7 is tested w.r.t. the accuracy $\epsilon_{te} > 0$ of the timing engine (in our experiments $\epsilon_{te} = 0.1\ ps$) and refuses a solution x' if it is not better than x by at least ϵ_{te}. In our experiments we were using ϵ_{init} as one micron, i.e., less than the standard height. In case $\mathrm{s}(c,x)$ is differentiable and concave, Algorithm 1 will find a locally optimal solution x_c up to an error of ϵ_{init}. Recall that $\mathrm{s}(c,x)$ may not be concave in x_c and, in general, Algorithm 1 may get stuck in a suboptimal solution.

Algorithm 1 Supergradient Ascent

1: *Input:* cell c, location vector x, initial step size ϵ_{init}, accuracy of timing engine ϵ_{te}
2: $\epsilon := \epsilon_{\text{init}}$.
3: **repeat**
4: Compute ascent direction $d_c(x)$ as in (2).
5: $x' := x$.
6: $x'_c := x'_c + \epsilon \cdot d_c$.
7: **if** $\mathrm{s}(c, x') \geq \mathrm{s}(c, x) + \epsilon_{te}$ **then**
8: $x := x', \epsilon := 2\epsilon$
9: **else**
10: $\epsilon := 0.5\epsilon$
11: **until** $\epsilon < \epsilon_{\text{init}}$

(a) Inverters are locally optimal.

(b) Cell c minimizes l_2-length.

Figure 2: Path straightening to escape poor solutions.

3.2 Path Straightening

Supergradient search is known to potentially show slow progress. We combined it with various line search variants proposed in text books, but found the simple scheme of Algorithm 1 superior on our instances. To speed up local search we are using a geometric idea, motivated by the example in Figure 2. Assume that the two outer cells have a fixed location. Then the placement of each upper cell is locally optimal, because each is placed on the shortest l_1-path between predecessor and successor. Algorithm 1 would leave them unchanged as a consequence of respecting the accuracy of timing data in line 7. But obviously moving both upper cells downward would result in a shorter path. For $\epsilon_{te} = 0$, under a linear delay model and perfect accuracy, the two inverters would move downwards, but only in many very small steps.

This motivates the following path straightening algorithm: It iteratively takes a cell $c \in \mathcal{C}$ and moves it to the center of the line segment connecting the critical upstream pin p_{up} and downstream pin p_{down} of c. Then it performs a line search for the maximum local slack $\mathrm{s}(c, x)$ along this segment in each direction, thereby respecting pin offsets so that the center of the critical input and output pin locations $x_{p_{in}}$ and $x_{p_{out}}$ of c is located on the segment between p_{up} and p_{down}. A solution minimizes the Euclidean path length between p_{up} and p_{down} (up to the offset of the pins of c).

Due to the iterative application, this algorithm straightens a critical path and leads to a fast convergence in the above example. It is summarized in Algorithm 2.

Of course, the line segment need not contain good solutions. Thus we consider Algorithm 2 as a fast preprocessing.

Algorithm 2 Path straightening

Input: locations x, cell c, initial step size ϵ_{init}
Let $x_{p_{up}}$, $x_{p_{down}}$, $x_{p_{in}}$ and $x_{p_{out}}$ denote the locations of the critical upstream, critical downstream pin, critical input pin, critical output pin, respectively.
Set $d_c := x_{p_{down}} - x_{p_{up}}$.
Set $z_c := \frac{1}{2}(x_{p_{up}} + x_{p_{down}})$.
Set $z'_c := \frac{1}{2}(x_{p_{in}} + x_{p_{out}})$.
for $\epsilon \in \{-\epsilon_{\text{init}}, \epsilon_{\text{init}}\}$ **do**
  repeat
    $\overline{x}_{c'} := \begin{cases} z_c + (x_c - z'_c) + \epsilon \cdot d_c & \text{if } c' = c, \\ x_{c'} & \text{if } c' \in \mathcal{C} - c. \end{cases}$
    if $\mathrm{s}(c, \overline{x}) > \mathrm{s}(c, x) + \epsilon_{te}$ **then**
      $x_c := \overline{x}_c, \epsilon := 2 \cdot \epsilon$
    else
      $\epsilon := 0.5 \cdot \epsilon$
  until step size is too small
Keep the best location among the initial location and those on the line segment.

3.3 Clustered Cell Movement

As shown in the introduction and Figure 1, our main goal of a clustered cell movement is to escape from suboptimal solutions trapping a single move search, because two or more gates are mutually blocking any progress. In Figure 1, we can escape the two situations by moving two or three gates simultaneously. In addition, a clustered move leads to a faster convergence in the right situation of Figure 1, if all interconnects have the same resistance. Any algorithm based on single cell movements would need many iterations to move the three cells down, which can be achieved by a single clustered move.

We cluster neighboring cells with a similar local slack in a single virtual cell, to which we apply Algorithm 1 and 2. Our clusters reflect both physical and slack-wise proximity. We compute them as follows: Let $\pi, \theta \in \mathbb{R}_+$ be a physical proximity and a slack proximity threshold, respectively.

Definition 1 *Given a cell $c \in \mathcal{C}$, we call a cell $c' \in \mathcal{C}$ (π, θ)-attracted to c, if c' and c are connected by a path s.t.*

- *for each pin on the path, its slack is within a θ-slack window around $\mathrm{s}(c, x)$: $[\mathrm{s}(c, x) - \theta, \mathrm{s}(c, x) + \theta]$ (slack attraction), and*
- *for every source/sink pair of pins p, p' in a net on the path, their distance is not greater than π (physical attraction).*

To form a cluster $C \subset \mathcal{C}$, we consider a cell $c \in \mathcal{C}$ and we add to C all cells that are (π, θ)-attracted to c. Note that we do not bound the physical radius of C, which in theory can be large as the connecting paths in C could contain many nets. However, we observed in practice that a cluster usually contains only a few cells.

Figure 3: Clustered Cell Movement

We identify C as a virtual cell with input pins and output pins for which we can define, for a placement x, the slacks $\mathrm{s}^-(C,x)$, $\mathrm{s}^+(C,x)$, and $\mathrm{s}(C,x)$ as for a single cell in (1). Fixing relative distances inside C, we can apply Algorithms 1 and 2 for the virtual cell C.

An example of a cluster and its movement is depicted in Figure 3. It shows the critical path in red and, in purple, a slightly less critical path within the θ-slack window. Assume that the purple inverter has a weak drive strength. Then it is important to cluster the purple inverter together with the red cells. Otherwise, the purple path would prevent a substantial movement of the red cells. In our experiments, we choose $\theta = 1ps$ and π to be twice the height of a standard cell. Empirically, we did not see instances were larger attractions provided a benefit.

3.4 Iterative Local Search

We now present how the local search algorithms are integrated into a loop for timing-driven placement refinement (Algorithm 3). The outer loop is iterated until no significant improvement of the worst slack is found ($\epsilon_{ws} = 0.5\ ps$ in our experiments). Inside the loop, we perform four major steps. First and second, the path straightening algorithm and the supergradient ascent are applied to single critical cells. Third and last, the same algorithms are executed also for critical cell clusters. As clustered cell moves take more running time because of the larger number of timing evaluations, we try to exploit the potential of single moves first.

Critical cells $C^\star$ are selected as follows. First, we traverse all nets by increasing slack at their source pins and select all cells that are attached to the current net and to all nets that have the same slack at their sources. As soon as more than $K \in \mathbb{N}$ cells are selected, the traversal of the nets stops. Note that this procedure selects the cells on the most critical paths and their direct successors for any choice of $K \geq 1$. In our experiments, we choose $K = |\mathcal{C}|/1000$. The critical cells are processed in reverse topological order (experiments with other process orders gave similar results). Cells that are initially uncritical might become critical and be considered after several iterations of the repeat-until loop. At the end of each loop iteration, before measuring the slack improvement, we legalize the cells minimizing quadratic movement [2].

It is important to select not only critical cells but all cells in their fanout because they can be moved to reduce the capacitance and length of the critical nets. Those cells are not selected if a delay model is used where less critical cells are not relevant for the delay, e.g. linear distance based wire delays. Furthermore, lines 12 and 17 ensure that no cell is added to multiple clusters in one step.

Algorithm 3 Refine Placement Loop

```
1: repeat
2:    Select critical cells C*
3:    for each cell in C* do
4:       Apply Algorithm 2 to c.
5:    Select critical cells C*
6:    for each cell in C* do
7:       Apply Algorithm 1 to c.
8:    Select critical cells C*
9:    for each cell in C* do
10:      Compute a cluster C as described in Section 3.3.
11:      Apply Algorithm 2 to C.
12:      C* := C* \ C
13:   Select critical cells C*
14:   for each cell in C* do
15:      Compute a cluster C as described in Section 3.3.
16:      Apply Algorithm 1 to C.
17:      C* := C* \ C
18:   Legalize placement with minimum perturbation [2].
19: until the worst slack improvement is below a threshold ε_ws.
```

3.5 Legalization and Congestion Mitigation

So far, we disregarded placement feasibility in Algorithms 1 and 2. To avoid large disturbances in line 18 caused by cells (cluster) which were moved on placement blockages, we consider blockages when evaluating the final solution found by the local search in Algorithms 1 and 2: The cell (cluster) is tentatively moved to the closest unblocked location, which is accepted as final solution if the local slack has improved compared to the starting solution.

Algorithm 3 with the infrequent legalization is not the only application scenario of our local search. Our program without clustering has been used as RefinePlace in the self-stabilizing framework of BonnPlace [3]. For fine tuning, an incremental placer could legalize individual cells immediately after their refinement as in [21].

Wire length and routing congestion may increase as a side effect of our local search algorithms. Typically, this effect is marginal since we consider only a small fraction of the cells. However, it is easily possible to integrate in our local search global routing calls similar to placement legalization.

3.6 Capacitance & Slew Limits

To avoid or reduce load capacitance and slew violations by our local search algorithms, we give them priority over the local slack. In fact, we are considering a lexicographic objective that minimizes 1) the total load violation, 2) the total slew violation, and 3)

the local slack, while searching for a better solution. Thus we are not worsening load and slew violations compared to the starting solution. To avoid degradation of the worst design slack, we refuse a new location at the end of a local search if it degrades the local slack compared to the starting solution.

When computing the ascent directions in Section 3.1, we relax the lexicographic objective function using fixed coefficients $\lambda_{cap}, \lambda_{slew} \in \mathbb{R}_+$ to penalize load and slew limit violations. Let tcap denote the total capacitance violation and tslew the total slew violation among all nets attached to the current cell (cluster), we estimate the supergradient of the following Lagrange function $s'(c, x) := (s(c, x) - \lambda_{cap} \cdot \text{tcap} - \lambda_{slew} \cdot \text{tslew})$. Since we focus rather on minimizing capacitance violations than on minimizing slew violations, we choose λ_{cap} to be 1024 and λ_{slew} to be 256, where slacks and slews are measured in ps and capacitances in fF in our experiments.

4 Experimental Results

We implemented the new local search algorithms in C++ and integrated them into an industrial design environment, using the sign-off timing engine EinsTimer from IBM. The industrial design flow runs in several steps with increasing accuracy of the delay model. The three stages use the following wire delay models:

1. Linear wire delays based on source-sink distances and constant gate delays (timing-driven global placement),
2. Elmore delays [9] (global delay optimization), and
3. AWE as in [19] (local delay refinement).

The timing-driven placement in the linear stage implements a netweighting scheme that iteratively increases weights on the critical paths as in [23]. In addition, long interconnects are assigned to wider wiring layers. In the Elmore and AWE stage, buffering, gate sizing, layer assignment, logic restructuring, and single cell moves are performed. Thus after each stage, the cell placement can be considered as well tuned w.r.t. timing constraints.

Now, we use the new local search at the end of each stage in the given delay model to post-optimize the placement. Our main testbed consists of 6 microprocessor units in 22nm technology with 23–459 thousand cells and clock frequencies between 4.8 and 5.75 GHz. The experiments were executed on a Linux cluster with Intel Xeon CPUs of clock frequencies between 2.9–3.4 GHz.

As we only move a small fraction of the cells ($K = |\mathcal{C}|/1000$), no substantial increase in wirelength is to be expected. To support this statement empirically, we measured routability before and after our local search using the global router that is integrated in the design environment.

Tables 1, 2 and 3 show the results of the new local search at the end of each stage. Note that linear wire delays are lower bounds for the achievable wire delay and slacks can be better in Table 1 compared to Table 3. The columns show the unit names, the number of cells $|\mathcal{C}|$ in thousands, the cycle time T, the worst late slack **WS** in picoseconds, the worst slack improvement **ΔWS** in percentage of the cycle time, total negative late slack

TNS in nanoseconds, total global routing overflow **OF**, the global routing wirelength **WL** in meters, and the running time **RT** in seconds.

Table 1: After stage 1 (linear wire delay)

	Unit		**End of Stage**				**Local Search Placement**					
	$\|\mathcal{C}\|$	T	**WS**	**TNS**	**OF**	**WL**	**WS**	**ΔWS**	**TNS**	**OF**	**WL**	**RT**
	K	ps	ps	ns		m	ps	%	ns		m	s
U1	82	174	-70	-41	0	1.65	**-65**	2.9	-40	0	1.65	60
U2	20	174	-173	-152	450	0.55	**-160**	7.5	-142	450	0.55	16
U3	112	174	-47	-29	0	2.30	**-41**	3.4	-24	0	2.30	151
U4	35	174	-64	-9	0	1.48	**-45**	10.9	-9	0	1.48	26
U5	116	174	-60	-16	0	3.15	**-9**	29.3	-5	0	3.15	435
U6	369	208	-81	-11	0	11.89	**-58**	11.1	-6	0	11.89	670
Avg.								**10.8**				

Table 2: After stage 2 (Elmore)

	Unit		**End of Stage**				**Local Search Placement**					
	$\|\mathcal{C}\|$	T	**WS**	**TNS**	**OF**	**WL**	**WS**	**ΔWS**	**TNS**	**OF**	**WL**	**RT**
	K	ps	ps	ns		m	ps	%	ns		m	s
U1	88	174	-105	-261	0	1.68	**-100**	2.9	-257	0	1.68	85
U2	21	174	-236	-266	96	0.57	**-173**	36.2	-261	96	0.57	67
U3	120	174	-87	-251	0	2.40	**-78**	5.2	-252	0	2.40	107
U4	55	174	-136	-279	0	1.92	**-128**	4.6	-278	0	1.92	122
U5	133	174	-139	-389	0	3.52	**-123**	9.2	-382	0	3.52	339
U6	433	208	-158	-762	0	12.60	**-153**	2.4	-748	0	12.60	473
Avg.								**10.1**				

Table 3: After stage 3 (AWE)

	Unit		**End of Stage**				**Local Search Placement**					
	$\|\mathcal{C}\|$	T	**WS**	**TNS**	**OF**	**WL**	**WS**	**ΔWS**	**TNS**	**OF**	**WL**	**RT**
	K	ps	ps	ns		m	ps	%	ns		m	s
U1	92	174	-75	-141	0	1.70	**-73**	1.1	-144	0	1.70	222
U2	23	174	-157	-213	80	0.57	**-149**	4.6	-211	80	0.57	111
U3	119	174	-47	-148	0	2.39	**-44**	1.7	-149	0	2.40	182
U4	62	174	-85	-135	0	1.99	**-81**	2.3	-133	0	1.99	1115
U5	140	174	-82	-128	0	3.56	**-72**	5.7	-125	0	3.56	1742
U6	458	208	-79	-205	0	12.65	**-77**	1.0	-208	0	12.65	1215
Avg.								**2.7**				

The average worst slack improvement is more than 10% in the first two stages and still 2.7 % in the AWE stage. Note that on one instance the improvement is 36%. There is only a slight increase of less than one percent in wirelength. The routing congestion does not increase.

To demonstrate the effect of the path straightening and the clustering in particular, we carried out further experiments to compare the following local search variants in the linear delay model of stage 1 on the same input placement:

1. supergradient ascent without clustering ("Supergradient"),
2. path straightening and supergradient ascent without clustering ("Straighten"), and

Table 4: Comparison of Local Search by Gradient, Path Straightening, and Clustering.

		Init.	Supergradient			Straighten			Cluster		
Unit	T	**WS**	**WS**		**RT**	**WS**		**RT**	**WS**		**RT**
	ps	ps	ps	Δ%	s	ps	Δ%	s	ps	Δ%	s
U1	174	-76	-72	2.3	36	-70	3.4	53	**-62**	8.0	130
U2	174	-199	-174	14.4	8	-168	17.8	13	**-155**	25.3	28
U3	174	-63	-32	17.8	46	-29	19.5	72	**-29**	19.5	142
U4	174	-76	-62	8.0	58	-60	9.2	76	**-38**	21.8	143
U5	174	-51	-41	5.7	96	-39	6.9	97	**-14**	21.3	466
U6	208	-68	-60	3.8	161	-57	5.3	304	**-43**	12.0	1085
Avg.				**8.7**			**10.4**			**18.0**	

3. path straightening, supergradient ascent, and clustering ("Cluster").

For each variant, we iterated on critical cells $\mathcal{C}^\star$ as in Algorithm 3, omitting the legalization step, until the worst slack improvement fell below $\epsilon_{ws} = 0.5ps$.

The common input placement arises from a wirelength-driven placement, followed by a congestion aware layer assignment. Table 4 shows the worst slack after the initial placement, and after each of the three variants, as well as the running times (RT).

The path straightening constitutes an improvement over the supergradient search, which stops earlier when the worst slack improvement falls below the improvement threshold ϵ_{ws}. The clustered moves yield a large gain. The average worst slack improvement increases from 10.4% to 18% compared to the path straightening method. The effect can be seen in Figure 4 showing a small part of U6. Here, the initial critical path persists after each local search variant. It starts at a common port on the left boundary and ends at a register in the right, which is pulled towards the center of the unit in the upper right by outgoing paths. Interconnects are drawn as l_2-segments of four different widths that are reflecting their relative width due to the layer assignment. Only the clustering variant is capable of moving the receiving latch and its neighbors jointly over the big macro block.

4.1 ICCAD benchmarks

For the sake of completeness, we conducted experiments on the instances of the "ICCAD 2014 Contest" on "Incremental Timing-driven Placement" [12]. These instances have the unrealistic setup that clock nets with up to 149,381 sinks are estimated by a short Steiner tree. A short Steiner tree topology with so many sinks tends to have a huge clock skew, and the topology as well as the skew are very sensitive to small moves of registers. Such a setup will hardly be found in practice. Instead, real designs are implementing a balanced and buffered clock tree. Before its construction, an idealized bounded skew clock tree is assumed during timing-driven placement and initial physical design. Thus, in this important aspect the contest setup is quite unrealistic. The probably best contest strategy is to reduce the clock schedule by small changes to the register locations in the hope that the Steiner tree accidentally reconnects high latency sinks in a more favorable way. In fact, the winning team added an extra step for tuning the skews [15].

Tuning the clock skew requires to use precisely the same extraction method as the evaluation script, up to the order in which sinks are stored at each net. As it is beyond

Table 5: Results on ICCAD benchmarks

Chip	$\|\mathcal{C}\|$	Input				Ours (with fixed registers)				Contest Winner (UFRGS/FURG BRAZIL)				
		WS	TNS	Skew	ABU	WS	TNS	ABU	Time	WS	TNS	Skew	ABU	Time
		ps	ns	ps		ps	ns		s	ps	ns	ps		s
b19	219	-1164	-16	2198	0.03	-255	-1	0.03	1076	**-109**	-0.2	1380	0.03	51
vga_lcd	164	-1333	-638	2618	0.01	-495	-179	0.07	1486	**-249**	-2	1886	0.00	92
leon3mp	649	-5610	-5990	40500	0.01	-4606	-5675	0.01	3161	**0**	0	31625	0.00	2649
leon2	794	-9702	-12636	72779	0.02	-8558	-9667	0.03	4768	**0**	0	47728	0.00	1086
netcard	958	-4511	-362	62660	0.01	-2811	-192	0.01	3779	**0**	0	45606	0.00	8044
mgc_edit_dist	130	-808	-94	888	0.00	**0**	0	0.00	587	-702	-34	545	0.00	4942
mgc_matrix_mult	155	-441	-3	266	0.00	**0**	0	0.00	196	**0**	0	217	0.00	2353

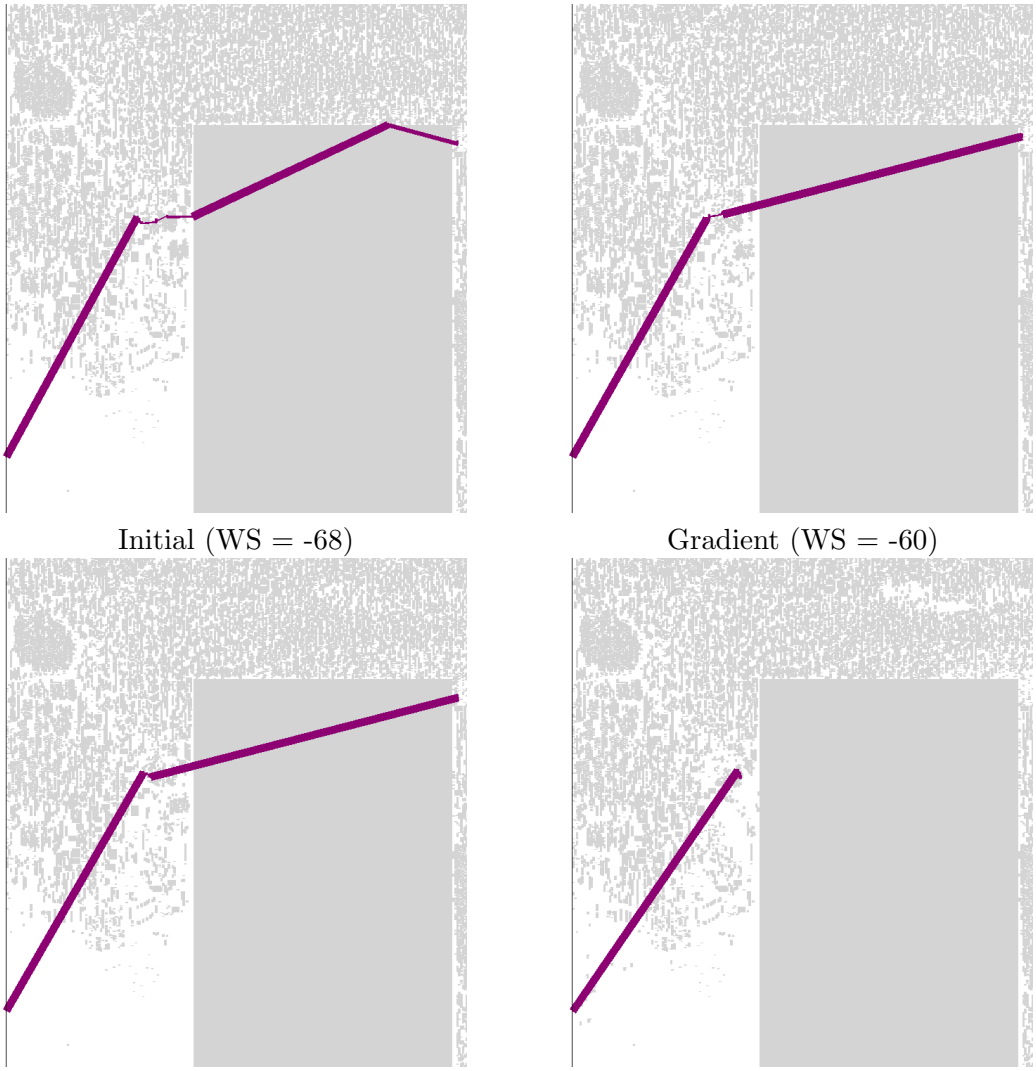

Figure 4: Critical paths on U6 using the 3 search methods.

our scope to integrate the contest extraction into the EinsTimer timing engine, we fixed all register positions and used the initial clock arrival times and slews that we read from the official evaluation script. This allows us to optimize at least the placement of the data paths, where we observed only small deviation in the Steiner extractions.

Table 5 shows the results on the contest instances using the "long" displacement constraints. The columns contain the instance name, the worst negative late slack (WS), the total negative late slack (TNS), clock skews (Skew), placement density deviation (ABU), and running times. The contest winner runs were performed on an Intel E5-2650 processor. Our results were evaluated with the official contest evaluation script and we obeyed the "long" move limits. Note that even without moving registers, we were

able to significantly reduce the worst late slacks and total negative slacks. Further slack improvement is mostly inhibited by the limit on the maximum move distance. For mgc_edit_dist, we can even close timing. The contest winners were extremely successful in reducing the clock skew, which we left unchanged. Their skew gain was significantly larger than the improvements in worst slack.

References

[1] A. H. Ajami and M. Pedram. "Post-Layout Timing-Driven Cell Placement Using an Accurate Net Length Model with Movable Steiner Points." ASPDAC, 595–600, 2001.

[2] U. Brenner. "VLSI Legalization with Minimum Perturbation by Iterative Augmentation." Automation and Test in Europe, 1385–1390, 2012.

[3] U. Brenner, A. Hermann, N. Hoppmann, and P. Ochsendorf. "BonnPlace: A Self-Stabilizing Placement Framework." ISPD, 9–16, 2015.

[4] W. Choi and K. Bazargan. "Incremental Placement for Timing Optimization." ICCAD, 463–466, 2003.

[5] 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?" DAC, 801–806, 2005.

[6] J. Cong, J. R. Shinnerl, M. Xie, T. Kong, and Xin Yuan. "Large-scale circuit placement." Transactions on Design Automation of Electronic Systems 10(2), 389–430, 2005.

[7] A. E. Dunlop, V. D. Agrawal, D. N. Deutsch, M. F. Jukl, P. Kozak, and M. Wiesel. "Chip Layout Optimization Using Critical Path Weighting." DAC, 133–136, 1984.

[8] S. Dutt and H. Ren. "Discretized Network Flow Techniques for Timing and Wire-Length Driven Incremental Placement With White-Space Satisfaction." Transactions on Very Large Scale Integration Systems 19(7), 1277–1290, 2011.

[9] W. C. Elmore. "The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers." Journal of Applied Physics 19(1), 55–64, 1948.

[10] S. Held. "Gate sizing for large cell-based designs." DATE, 827–832, 2009.

[11] M. A. B. Jackson and E. S. Kuh. "Performance-Driven Placement of Cell Based IC's." DAC, 370–375, 1989.

[12] M.-C. Kim, J. Hu, and N. Viswanathan. "ICCAD-2014 CAD Contest in Incremental Timing-Driven Placement and Benchmark Suite." ICCAD, 361-366, 2014.

[13] T. Kong. "A Novel Net Weighting Algorithm for Timing-Driven Placement." ICCAD, 172–176, 2002.

[14] T. Luo, D. A. Papa, Z. Li, C.N. Sze, C.J. Alpert, and D.Z. Pan. "Pyramids: An Efficient Computational Geometry-based Approach for Timing-driven Placement." ICCAD, 204–211, 2008.

[15] J. Monteiro, G. Flach, J. C. Puget, M. P. Fogaca, P. F. Butzen, and M. de Oliveira Johann. Personal communication, 2015.

[16] B. Obermeier and F. M. Johannes. "Quadratic Placement Using an Improved Timing Model." DAC, 705–710, 2004.

[17] D. A. Papa, T. Luo, M. D. Moffitt, C. N. Sze, Z. Li, G.-J. Nam, C. J. Alpert, and I. L. Markov. "RUMBLE: An Incremental, Timing-driven, Physical-synthesis Optimization Algorithm." ISPD, 2–9, 2008.

[18] K. Rajagopal, T. Shaked, Y. Parasuram, T. Cao, A. Chowdhary, and B. Halpin. "Timing Driven Force Directed Placement with Physical Net Constraints." ISPD, 60–66, 2003.

[19] C. Ratzlaff and L. T. Pillage. "RICE: Rapid Interconnect Circuit Evaluation Using Asymptotic Waveform Evaluation." Transactions on Computer-Aided Design 13(6), 763–776, 1994.

[20] H. Ren, D. Z. Pan, and D. S. Kung. "Sensitivity Guided Net Weighting for Placement Driven Synthesis." Transactions on Computer-Aided Design of Integrated Circuits and Systems 24, 711–721, 2005.

[21] L. Trevillyan, D. S. Kung, R. Puri, L. N. Reddy, and M. A. Kazda. "An Integrated Environment for Technology Closure of Deep-Submicron IC Designs." Design & Test of Computers 21(1), 14–22, 2004.

[22] R.-S. Tsay and J. Koehl. "An Analytic Net Weighting Approach for Performance Optimization in Circuit Placement." DAC, 620–625, 1991.

[23] N. Viswanathan, G.-J. Nam, J.A. Roy, Z. Li, C.J. Alpert, S. Ramji, and C. Chu. "ITOP: Integrating Timing Optimization within Placement." ISPD, 83–90, 2010.

[24] Q. Wang, J. Lillis, S. Sanyal. " An LP-based Methodology for Improved Timing-Driven Placement." ASPDAC, 1139–1143, 2005.