# A Linear Time Algorithm for Optimum Tree Placement

Satrajit Chatterjee Zile Wei Alan Mishchenko Robert Brayton

Department of EECS
U. C. Berkeley
{satrajit, zile, alanmi, brayton}@eecs.berkeley.edu

## ABSTRACT

Let $N$ be a point placed directed graph and $G$ be a sub-graph of $N$. We study the point placement problem of optimally re-placing the nodes in $G$ assuming that the remaining nodes in $N$ are fixed. The goal is to minimize the total "wiring" cost i.e. the cost of the internal nets of $G$ plus the external nets connecting $G$ to other other nodes of $N$. The cost of a net is the semi-perimeter of the smallest bounding box of the nodes of the net. The main result of this paper is a linear time algorithm (DP) for this problem that is *optimum* when $G$ is a tree; nodes of $G$ can have multiple fanouts, but for each net in $G$ there can be no more than one fanout to other nodes of $G$. Experiments verify the optimality of the algorithm by comparisons with a linear programming (LP) formulation of the problem. The algorithm is much faster (linear) than the LP (quadratic) in practice, and for non-trees was only marginally sub-optimal in the application tested. Several application areas are suggested, one being placement-aware logic synthesis.

## 1. INTRODUCTION

Let $N$ be a directed graph and $G$ be a sub-graph of $N$. We assume that the nodes of $N$ are given point placements in the plane. We study the problem of optimally re-placing the nodes in $G$ assuming that the remaining nodes in $N$ are fixed at their given placements. A net consists a source node plus all sink nodes (fanouts) of edges that have the same source. The cost of a net is taken to be the semi-perimeter of its smallest bounding box of the nodes of the net. The goal is to place $G$ such that the total "wiring" cost of $N$ is minimized i.e. minimize the cost of the internal nets of $G$ plus the cost of the external nets connecting $G$ to the fixed nodes.

The DP algorithm presented here can find the optimum placement of $G$ in linear time if

1. $G$ is a tree, and
2. any node not in $G$ fans in to at most one node in $G$.

Note that any node of $G$ may fanout to any number of nodes not in $G$, i.e. the fixed nodes. Although optimality is not guaranteed for DAGs, the algorithm can be used as a heuristic to place DAG sub-graphs; for families of small DAGs, experimental comparisons show that the algorithm is only marginally sub-optimal.

The general placement problem of this type can be formulated and solved for DAGs as a linear program. However, our proposed algorithm runs in linear time and in practice is much faster than sparse LP (which experimentally runs in quadratic time). Speed is important in applications where the method may be used to evaluate many candidate restructurings of a netlist.

In Section 2 we review the optimum placement algorithm when the sub-network is a single node. Section 3 describes the tree placement algorithm and Section 4 discusses prior work related to tree placement in the literature. Section 5 presents some possible applications of the proposed algorithm, and Section 6 presents experiments comparing the proposed algorithm with a linear programming formulation for speed and optimality. A proof of optimality is given in the appendix.

## 2. THE NODE PLACEMENT PROBLEM

### 2.1 The Problem

Let $p$ be a point in the plane, $r$ a given rectangle, $[r, p]$ the smallest rectangle containing $r$ and $p$, and $\text{cost}(r)$ be the semi-perimeter of rectangle $r$.

In the node placement problem, we are given a set of rectangles $R = \{r_1, r_2, .., r_k\}$, and we want to choose a point $p$ such that $\sum_{r=1}^{k} \text{cost}([r_i, p]) - \text{cost}(r_i)$ is minimized.

The node placement problem was formulated as an abstraction of the problem of optimally placing a new gate $g$ in a logic network whose nodes are already placed. The rectangles $r_i$ represent the bounding boxes of nets that will be connected to $g$, either as inputs or output. The optimal solution of the node placement problem provides the optimal location of $g$ that minimizes the total wire length (measured as the semi-perimeter of the bounding boxes of nets) of the network when $g$ is added to the network.

This problem was considered in [10, 4] and the *median* algorithm was presented to solve it. We summarize the derivation of that algorithm from [4] here for reference.

Let $p^x$ represent the projection of a point $p$ onto the x-axis. Let $r^x$ represent the segment obtained by projecting rectangle $r$ onto the x-axis. Similarly for the y-axis. Now observe that,

$$\text{cost}([r, p]) = \text{cost}([r^x, p^x]) + \text{cost}([r^y, p^y])$$

where $[r^x, p^x]$ represents the bounding segment of $r^x$ and $p^x$, and the cost of a segment $r$ is its length. Therefore, the node placement problem separates into two 1-dimensional problems which can be solved independently. We describe the procedure to solve the 1-dimensional problem in the next section.

## 2.2 The Median Algorithm

In the one dimensional problem, we have a set of segments $R = \{r_1, r_2, .., r_k\}$. If $R$ is a set of segments, define $\phi_R(p) \equiv \sum_{r \in R} \mathrm{cost}([r,p]) - \mathrm{cost}(r)$. Our goal then is to pick a point $p$ to minimize $\phi_R(p)$. We refer to such a point $p$ as an optimal solution of the node placement problem $R$.

For the formal treatment we assume that the end-points of $r_i$ are all distinct. This is without loss of generality since a segment can always be perturbed slightly without increasing $\phi(p)$.

Let $\mathcal{N}_l(p, R)$ be the segments $r_i \in R$ that lie entirely to the left of $p$. $\mathcal{N}_l(p, R)$ is shortened as $\mathcal{N}_\mathrm{l}(p)$ if $R$ is unambiguous in context. Similarly $\mathcal{N}_\mathrm{r}(p)$ for right. Let $\mathcal{N}_\mathrm{o}(p)$ be the segments that contain $p$. Note that $\mathrm{cost}(r,p) - \mathrm{cost}(r) = 0$ for all $r \in \mathcal{N}_\mathrm{o}(p)$. Therefore, $\phi(p)$ is equal to

$$\sum_{r \in \mathcal{N}_\mathrm{l}(p)} \Big(\mathrm{cost}([r,p]) - \mathrm{cost}(r)\Big) + \sum_{r \in \mathcal{N}_\mathrm{r}(p)} \Big(\mathrm{cost}([r,p]) - \mathrm{cost}(r)\Big)$$

Furthermore, if $r_\mathrm{l}$ and $r_\mathrm{r}$ are the end points of segment $r$, $r_\mathrm{l} < r_\mathrm{r}$, then we have

$$\phi(p) = \sum_{r \in \mathcal{N}_\mathrm{l}(p)} (r_\mathrm{r} - p) + \sum_{r \in \mathcal{N}_\mathrm{r}(p)} (p - r_\mathrm{l})$$

**Theorem 1** **([4]).** *$\hat{p}$ is optimal iff $|\mathcal{N}_\mathrm{l}(p)| = |\mathcal{N}_\mathrm{r}(p)|$.*

Proof. If $\hat{p}$ is optimal, then $|\mathcal{N}_\mathrm{l}(p)| = |\mathcal{N}_\mathrm{r}(p)|$. Otherwise, suppose $|\mathcal{N}_\mathrm{l}(p)| > |\mathcal{N}_\mathrm{r}(p)|$. Moving $\hat{p}$ to the left would reduce $\sum_{r \in \mathcal{N}_\mathrm{l}(p)} (r_\mathrm{r} - \hat{p})$ more than it would increase $\sum_{r \in \mathcal{N}_\mathrm{r}(p)} (\hat{p} - r_\mathrm{l})$ thus improving the solution.

Conversely, if $|\mathcal{N}_\mathrm{l}(p)| = |\mathcal{N}_\mathrm{r}(p)|$ then $\hat{p}$ is optimal since in this case in a neighborhood around $\hat{p}$, $\phi(p)$ simplifies to $\sum_{r \in \mathcal{N}_\mathrm{l}(p)} r_\mathrm{r} - \sum_{r \in \mathcal{N}_\mathrm{r}(p)} r_\mathrm{l}$ which does not depend on $p$. Thus $\hat{p}$ is a local optimum. Since $\phi(p)$ is convex, $\hat{p}$ must also be globally optimal. $\square$

The above observations lead to the median algorithm to optimally pick $p$.

1. sort the end-points of the $k$ segments $r_i$ into a sequence $x_1, x_2, .., x_{2k}$ such that $x_j < x_{j+1}$, $j = 1..(2k-1)$.

2. return the segment $[x_k, x_{k+1}]$.

The returned segment is a range of optimal values for $p$, and is called the *median* of the set of segments $R$, and is denoted by $\mathrm{med}(R)$.

For the 2-dimensional node placement problem $R$, we obtain two 1-dimensional medians (one for each axis). Together they define a rectangular region which is called the median of $R$.

## 3. THE TREE PLACEMENT PROBLEM

The tree placement problem is a generalization of the node placement problem. As before, we are given a collection of rectangles $R$ in the plane, but instead of a node, we are given a directed tree (defined below) of nodes to be connected to the rectangles. The problem is to find a placement for the tree that minimizes the total cost (which includes the internal connections of the tree as well as connections to the rectangles).

## 3.1 The Problem

Let $G$ be a directed acyclic graph with nodes $N$ and edges $E$ ($E \subseteq N \times N$). If $(u, v) \in E$ then $u$ is called an input of $v$, and $v$ an output of $u$. If every $n \in N$ has at most one output, and there is only one node with no output, then $G$ is called a directed tree. The node with no output is called the root. Nodes with no inputs are called leaves. Also we represent the set of inputs of a node $n$ by $\mathrm{input}(n)$, i.e. $\mathrm{input}(n) = \{m \mid (m, n) \in E\}$. Also, if $(u, v) \in E$, then $v$ is called the parent of $u$.

Let $T$ be a directed tree. Let $\rho$ be a function that maps nodes of $T$ to sets of rectangles. $\rho$ is called the input map because it assigns input nets to be inputs of nodes of $T$. The map $\rho$ is restricted to be tree-like in the sense that $\rho(n_1) \cap \rho(n_2) \neq \emptyset$. Thus inputs of $T$ cannot reconverge in $T$. Further, $\rho(n)$ for a leaf node of $T$ is not empty; a node must have at least one input.

Let $\tau$ be a function that maps each node of $T$ to a rectangle, which may be empty. $\tau$ is called the output map because it assigns a node of $T$ to fanout to a given rectangle. A node of $T$ may also fanout to another node of $T$ as well, but not more than one. Thus nodes of $T$ may have multiple fanouts, but at most one can be another node of $T$. We only need consider one rectangle to represent part of the output net of a node connecting all fanouts other than a fanout of $T$.

## 3.2 Tree Placement Algorithm (DP)

The algorithm (called the DP algorithm) is a simple two pass algorithm for optimally placing a tree shown as Procedure 1. The pseudo-code invokes the corresponding median computation (as described at the end of Section 2.2) as a subroutine.

The first pass is done in a bottom-up manner: We process each node $n$ in a topological order starting from the leaves. If $n$ is a leaf node, then $\mathrm{input}(n)$ is empty and $\mathrm{rect}[n]$ is the median of $\rho(n) \cup \tau(n)$. If $n$ is not a leaf node, then $\mathrm{rect}[n]$ is the median of $\rho(n)$, $\tau(n)$, and $[\tau(i), \mathrm{rect}[i]]$ for each input $i$ of $n$.

After the first pass, we decide on a location for the root node in its median region. (Any location within that region has the same cost.) Then the second pass is done in a top-down manner: We process all the nodes (except the root) in reverse topological order. We decide on the location of the parent of a node before processing the node. For each node $n$, we compute a new median region $R$ based on $\rho(n)$, the input rectangles $[\tau(i), \mathrm{input}(i)]$ ($i \in \mathrm{input}(n)$), and the expanded output rectangle $[\tau(n), P(\mathit{parent}(n))]$, which contains the location of the parent. We fix a position for $n$ in the new median region before moving on to the next node.

Note that the DP algorithm has some flexibility in placing a node in the median during the backward pass. A secondary cost function (such as density of points in a region) could be used to choose an appropriate point in the median.

**Remark.** The DP algorithm can be generalized to handle non-tree structures where a node in $G$ can have multiple parents. But then we can not guarantee the optimality for node placement procedure. However we observe from experiments that the procedure often finds an optimum solution even in a non-tree structure, and as a heuristic, it is reasonably accurate.

*Example* 1: Figure 1 shows how the DP algorithm generates the optimal locations for the $y$-dimension. The top-left of Figure 1 is a subnetwork with 3 nodes to be placed opti-

**Procedure 1** DP$(T, \rho, \tau)$

**Input:** A tree placement problem $(T, \rho, \tau)$
**Output:** A placement $P$ of $T$

```
for each node $n$ in $T$ in topological order do
    $R \leftarrow \tau(n) \cup \rho(n) \cup \{[\tau(i), \mathrm{rect}[i]] \mid i \in \mathrm{input}(n)\}$
    $\mathrm{rect}[n] \leftarrow \mathrm{med}(R)$
end for

$r \leftarrow \mathrm{root}(T)$
$P[r] \leftarrow$ any point in $\mathrm{rect}[r]$

for each node $n \neq r$ in $T$ in reverse topological order do
    $R \leftarrow \rho(n) \cup \{[\tau(i), \mathrm{rect}[i]] \mid i \in \mathrm{input}(n)\}$
    $R \leftarrow R \cup [\tau(n), [P(\mathrm{parent}(n))]]$
    $P[n] \leftarrow$ any point in $\mathrm{med}(R)$
end for
```

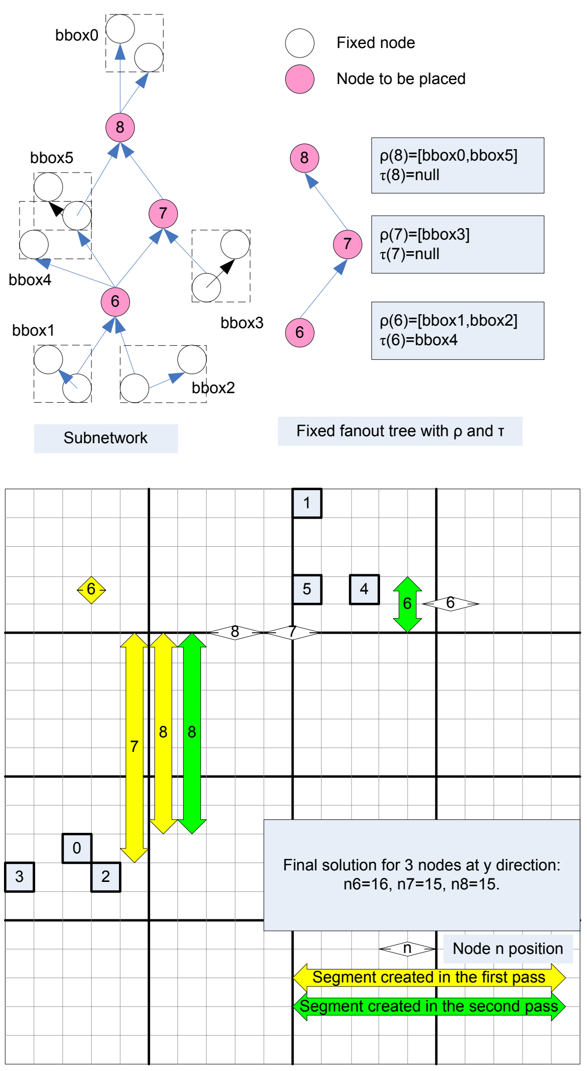

**Figure 1: The algorithm run on Example 1.**

mally. All other nodes are fixed. We compute the bounding box of fixed nets and build a placement structure in the top-right of Figure 1. $\rho$ and $\tau$ consist of bounding boxes of fixed nets. Note that $\tau$(node 7) and $\tau$(node 8) are empty. The location of the $y$-segments of the bounding boxes are shown in the bottom of Fig 1. All bounding boxes are unit-sized rectangles located on a $[0, 20] \times [0, 20]$ square. In the first pass, the segments for node 6, node 7 and node 8 are assigned according to $\mathrm{med}\big(\tau(n) \cup \rho(n) \cup \{[\tau(i), \mathrm{rect}[i]] \mid i \in \mathrm{input}(n)\}\big)$. In the second pass, we locate node 8 at 15 (from the segment [8,15]), and then, node 7 position has to be 15. We pick node 6 at 16 from the segment [15,17]. The minimal cost for placing these 3 nodes is 29.

## 3.3 Asymptotic Run-time

We assume that the median of $k$ points can be found in $O(k)$ time [6, Chapter 9]. From the pseudo-code shown in Procedure 1, we see that for a node $n$, every rectangle in $\tau(n)$, $\rho(n)$ and $\mathrm{rect}(n)$ participates in at most a constant number of median computations. Therefore, the run-time is linear in the size of the input problem i.e. linear in the size of the tree $T$ and the maps $\tau$ and $\rho$. (In practice, we do not use the linear-time median algorithm, but instead use sorting since the median problems are small.)

## 4. RELATED WORK

The problem of optimally placing a weighted tree given the locations of its leaf nodes was addressed by Fischer and Paterson [7]. They obtain a $O(n \log n)$ dynamic programming algorithm for the general case when the edges of the tree have arbitrary weights, and specialize it to obtain an $O(n)$ algorithm for binary trees with uniform weights. Their approach is to explicitly store the function that captures the cost of optimally placing a sub-tree given a location for the root node of the sub-tree.

The DP algorithm presented here has certain advantages over the algorithm of Fischer and Paterson. First, we do not assume fixed point locations for the leaves, but consider connections of the leaves and internal nodes to external rectangles (nets). (However, their method may be modified to use this cost function.) Second, the DP algorithm is significantly simpler in terms of implementation and proof of optimality. Further, since we do not maintain any expensive data structures (their method uses 2-3 trees, etc. to get the $O(n \log n)$ bound), the practical run-time is much faster. Also, our method is $O(n)$ for arbitrary (not just binary) trees.

This simplication is obtained at the cost of reducing generality. By insisting that all edge weights are the same, we exploit a peculiar property of the median algorithm. Given $k + 1$ segments for the node placement problem, the node can be optimally placed (roughly speaking) knowing only any $k$ of the segments. The proof of optimality presented in the Appendix uses this property to establish the correctness of the DP algorithm. In this connection, the problem of extending the DP algorithm to handle arbitrary weights (especially integral or rational) appears interesting.

Two other problems considered in the literature in connection with tree placement are the linear arrangement MINCUT and MINSUM problems (see [17] and the references there-in). These are concerned with finding the optimum 1-1 mappings from the set of nodes $N$ of a graph to the set of integers $\{1, 2, ..., |N|\}$ for some cost functions. However, these problems are qualitatively different from the point placement problem, since no “overlaps” are allowed. Consequently, these problems are NP-hard for general graphs though there are polynomial-time algorithms for trees.

We specifically mention the linear arrangement problems since they have been considered in the placement-aware technology mapping [13] to minimize the sum of cell area and routing channel area (the cells are constrained to be in a row). We believe that the DP algorithm could also be used with a technology mapping algorithm to optimize a combination of cell area and total wirelength which arguably is more important in modern over-the-cell routing schemes.

It was pointed out by a reviewer that the deferred merge embedding (DME) algorithm used in clock tree construction [3, 5] is similar to the DP algorithm presented here. However, clock tree construction is concerned with placing *binary* trees whose leaves have fixed locations (subject to some additional constraints on skew). The present work may be seen as a generalization of the DME idea to *arbitrary* trees whose leaves and internal nodes may be connected to external rectangles (nets).

## 5. APPLICATIONS

In general, the DP algorithm is a point placement method that can be considered as an alternative to analytic algorithms, such as quadratic placement, which approximates net length by using weighted Euclidian distances. The semi-perimeter bounding box is a better estimate of actual net length and for three point nets matches the Steiner tree length. Even though DP is not optimum for non-tree structures, its speed allows for iterative placement focused on improving the result at multiple fanout points where the DP method overestimates wirelength; each iteration is guaranteed to reduce total wire length. Large trees can be extracted from a DAG by choosing a single fanout for each node and letting the other fanouts be fixed.

Another application comes from placement-aware logic synthesis in the manner of Pedram and Bhat [15, 16]. Starting with an initial placement of a net list, one estimates the wiring cost of a potential modification of the logic by replacing only the affected sub-network. Many different modifications of each sub-network are evaluated, and the best one is chosen according to some criterion that incorporates wiring cost. Then the net list is modified accordingly and the placement is updated with the positions computed in the estimate. The process continues until some stopping criterion is met. This general approach has been applied successfully in a number of different settings to minimize congestion [4, 11, 12] and to improve timing [8, 9].

In the methods that have appeared in the literature so far, the sub-network to be placed usually has been restricted to a single node [4, 10]; obtaining good placements for multiple nodes possibly was deemed too computationally expensive for evaluating potential modifications since many have to be done.

The present work was motivated by the problem of extending re-writing based logic synthesis [14] to be placement-aware. One form of this type of re-writing examines several different logic structures for every 4-input cut of every node in an AIG representing the entire logic. Then the best structure that leads to maximum logic sharing with the rest of the network is chosen. To extend this to be placement-aware, we need to use a cost function which accounts for the effect on wire length. Since each rewriting structure examined, typically has multiple nodes, we need an algorithm to place them optimally all at once. Since every logic structure of every cut of every node needs to be examined, we need a extremely fast method, particularly since rewriting itself is extremely fast.

## 6. EXPERIMENTAL RESULTS

Experiments were done to verify the following properties of the DP algorithm:

1. It is optimum when applied to trees.
2. It is linear in practice versus sparse LP which is quadratic in practice.
3. Its non-optimality when applied to non-trees is small.

All experiments were run on a Linux server with Xeon 3.06GHz processor and 4G memory. We implemented the DP algorithm on the ABC [2] platform. The internal representation of an AIG node was modified to record an $(x, y)$ coordinate, which was pre-computed by a Gordian placer. Each AIG node was thus associated with a location in a 2D plane. We extracted small sub-networks associated with 4-input cuts of a node. The cut fanin and fanout nets had known locations. Nodes between a node and the cut were to be placed. Nodes in the cut boundary are fixed and included in either $\rho$ or $\tau$.

In order to obtain the exact optimum value and verify the optimality of the DP algorithm, we derived a standard linear programming (LP) formulation of the minimal half-perimeter wire length problem and employed the GLPK package [1] to find an optimum solution. The total number of variables and constraints in the LP was roughly 3X-4X the number of nets. Since all constraints have at most 3 non zero coefficients, we used the sparse LP routine in GLPK to find an optimum solution.

In addition to subnetworks extracted from *i10.blif*, one of MCNC benchmarks, we selected a list of tree/DAG circuits, shown in Figure 3, whose fanins and fanouts were randomly distributed inside a placement area. In particular, we were interested in three types of circuit structures: k-nary trees, k-nary trees with fixed pins, and DAGs with fanout nodes. We verified experimentally that the DP algorithm is optimum for the first two types of structures but was in general suboptimal for DAGs.

The two graphs at the top in Figure 3 show the run times of LP and the DP algorithm on complete binary trees. The $x$-axis corresponds to the number of leaves in the trees, ranging from 3 to 64 and the $y$-axis shows the runtimes. Given a binary tree, we assigned 1000 random positions for its fanins and fanouts. For LP, the runtime is the total CPU time for computing all 2000 linear programming problems (since one tree is separated into two LP problems which are solved independently). As expected, the proposed algorithm scales linearly. The bottom graph of Figure 3 shows the ratio of the runtime of LP and the DP algorithm. This ratio grows almost linearly, indicating that the complexity of the sparse LP linear programming algorithm is quadratic to the size of circuit.

Table 1 depicts the runtime of the two algorithm on several circuit structures: k-nary trees, the two DAGs in Figure 2 and 4-input subnetworks from *i10.blif*. $k$, $h$ and $f$ are the number of branches, height of tree, and the number of fixed fanout nets, respectively. $n$ and $m$ are the number of fanins and the number of unknown nodes. k-nary trees

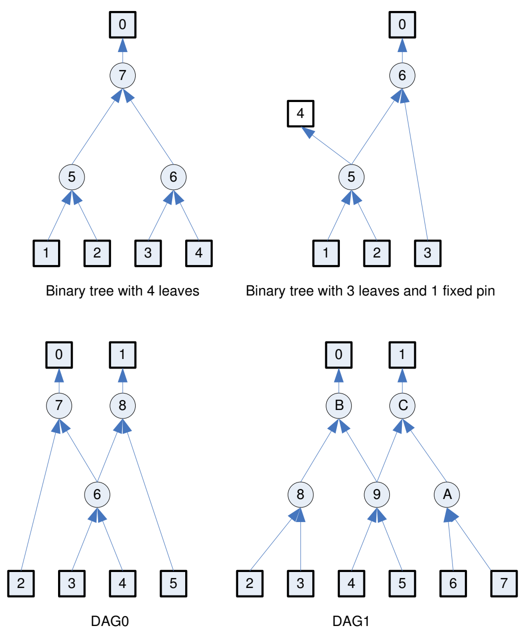
Binary tree with 4 leaves
Binary tree with 3 leaves and 1 fixed pin
DAG0
DAG1

**Figure 2: Circuit structures used for experiments.**

and subnetworks from a benchmark have only one fanout. DAG0 and DAG1 have 2 fanouts.

From Table 1, in all structures, the DP algorithm is at least 16 times faster than LP, and with more unknown nodes to be placed, this speedup is even more reflecting a quadratic complexity of LP. Surprisingly, about half of the random samples of DAG0 and DAG1 and 99.4% of the 4-input subnetworks from *i10.blif* are solved optimally, although our algorithm doesn't guarantee optimality for general DAG structures, such as, DAG0, DAG1 and general 4-input subnetworks. In those cases of non-optimality, the DP algorithm overestimates wire length 11.8% in DAG0, 7.9% in DAG1 and 11.7% in 4-input subnetworks.

## 7. CONCLUSIONS

We presented a fast and simple procedure to optimally place trees where internal nodes and nets may be connected to other fixed nodes. The algorithm can be run also on DAGs, and our experimental results show that it provides good placements (though not optimum) at least for small DAGs.

We note that the key feature enabling the algorithm is that it is possible to optimally place a node when all but one of its connections are fixed. The "directionality" of trees essentially captures this restriction. Therefore, we can imagine this algorithm being applied to quickly replace a large set of nodes in circuits which are not trees.

Also it will be interesting to see how the DP algorithm competes with other analytical placers which use Euclidian distance and quadratic programming for global placement. On the one hand, our placer in not optimum, but it does use a better metric for total wirelength. On DAGs, our placer tends to overestimate wirelength since it essentially overcounts each multiple fanout net. We will examine heuristics which suppress this aspect.

Finally, an interesting difference between the proposed al-

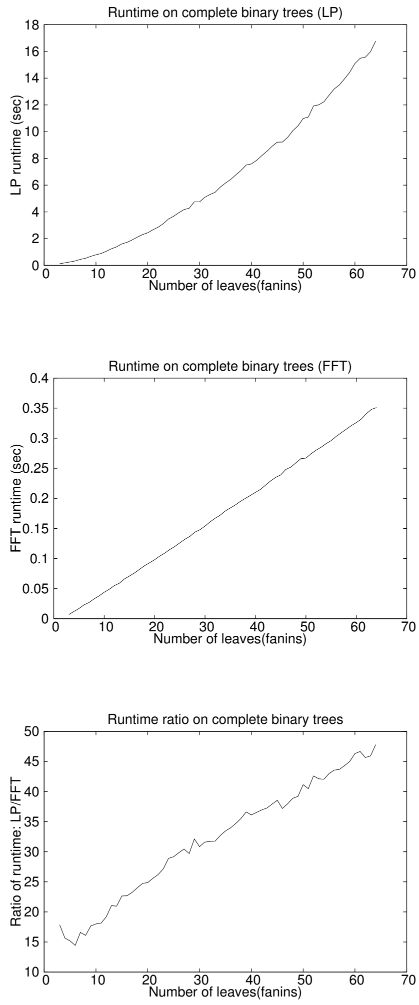

**Figure 3: Graphs showing runtime of LP (top), runtime of the DP algoritm (middle) and the ratio of the two.**

gorithm and the LP (or quadratic programming) formulations is that the entire graph need not be a fixed topology, before placement can begin. This suggests that this algorithm may be used in placement-aware technology mapping, etc. where the entire circuit structure has not yet been decided.

## 8. ACKNOWLEDGMENTS

This work was partly supported by SRC (# 1361.001 and # 1444.001) and the California Micro Program with our industrial sponsors Altera, Intel, Magma, and Synplicity.

## 9. REFERENCES

[1] "GNU Linear Programming Kit, Version 4.9," *http://www.gnu.org/software/glpk/glpk.html*

[2] Berkeley Logic Synthesis and Verification Group, "ABC: A system for sequential synthesis and verification, Release 61225," *http://www.eecs.berkeley.edu/~alanmi/abc/*

| structure | stat | | | | | runtime(sec) | | |
|---|---|---|---|---|---|---|---|---|
| | k | h | f | n | m | LP | DP | ratio |
| k-nary tree | 2 | 2 | 0 | 4 | 3 | 0.28 | 0.02 | 16.59 |
| | 2 | 3 | 0 | 8 | 7 | 0.68 | 0.04 | 18.03 |
| | 3 | 2 | 0 | 9 | 4 | 0.65 | 0.03 | 20.31 |
| | 3 | 3 | 0 | 27 | 13 | 3.55 | 0.11 | 32.25 |
| | 4 | 2 | 0 | 16 | 5 | 1.37 | 0.06 | 22.92 |
| | 4 | 3 | 0 | 64 | 21 | 13.42 | 0.26 | 52.25 |
| | 2 | 2 | 1 | 4 | 3 | 0.37 | 0.02 | 18.70 |
| | 2 | 3 | 1 | 8 | 7 | 0.82 | 0.04 | 19.45 |
| | 3 | 2 | 1 | 9 | 4 | 0.77 | 0.04 | 20.70 |
| | 3 | 3 | 1 | 27 | 13 | 3.79 | 0.11 | 33.58 |
| | 4 | 2 | 1 | 16 | 5 | 1.46 | 0.06 | 22.81 |
| | 4 | 3 | 1 | 64 | 21 | 13.42 | 0.26 | 51.41 |
| | 2 | 2 | 2 | 4 | 3 | 0.48 | 0.02 | 21.95 |
| | 2 | 3 | 2 | 8 | 7 | 0.92 | 0.04 | 20.56 |
| | 3 | 2 | 2 | 9 | 4 | 0.90 | 0.04 | 22.40 |
| | 3 | 3 | 2 | 27 | 13 | 4.19 | 0.12 | 35.82 |
| | 4 | 2 | 2 | 16 | 5 | 1.62 | 0.07 | 24.25 |
| | 4 | 3 | 2 | 64 | 21 | 14.12 | 0.27 | 53.09 |
| DAG0 | - | - | - | 4 | 3 | 0.34 | 0.02 | 17.10 |
| DAG1 | - | - | - | 6 | 5 | 0.57 | 0.03 | 18.55 |
| i10 | - | - | - | - | - | 2.49 | 0.15 | 16.50 |

**Table 1: Runtime on three types of circuit structures and sub-graphs of *i10*.**

[3] T.-H. Chao, Y.-C. Hsu, J.-M. Ho and A. B. Kahng, "Zero skew clock routing with minimum wirelength," In *IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing*, Vol. 39, No. 11, pp. 799-814, 1992

[4] S. Chatterjee and R. Brayton. "A new incremental placement algorithm and its application to congestion-Aware divisor extraction," In *Proc. of ICCAD,* pages 541-548, 2004.

[5] J. Cong, A. B. Kahng, C. K. Koh and C.-W. A. Tsao, "Bounded-Skew Clock and Steiner Routing", In *ACM Trans. on Design Automation of Electronic Systems,* Vol. 3, No. 3, pp. 341-388, 1998.

[6] T. Cormen et al. *Introduction to Algorithms,* 2nd Edition, MIT Press, 2002.

[7] M. Fischer and M. Paterson. "Optimal tree layout (Preliminary version)," In *Proc. of STOC,* pages 177-189, 1980. http://doi.acm.org/10.1145/800141.804665

[8] W. Gosti, A. Narayan, R. Brayton and A. Sangiovanni-Vincentelli "Wire-planning in logic synthesis," In *Proc. of ICCAD,* pages 26-33, 1998.

[9] W. Gosti, S. Khatri and A. Sangiovanni-Vincentelli. "Addressing the timing closure problem by integrating logic optimization and placement," In *Proc. of ICCAD,* pages 224-231, 2001.

[10] S. Goto "An efficient algorithm for the two-dimensional placement problem in electrical circuit layout." In *IEEE Transitions on Circuits and Systems,* Vol. CAS-28, No. 1, pages 12-18, 1981.

[11] T. Kutzschebauch and L. Stok. "Layout driven decomposition with congestion consideration," In *Proc. of DATE,* pages 672-676, 2002.

[12] T. Kutzschebauch and L. Stok. "Congestion aware layout driven logic synthesis," In *Proc. of ICCAD,* pages 216-223, 2001.

[13] J. Lou, A. Salek and M. Pedram. "An exact solution to simultaneous technology mapping and linear placement problem," In *Proc. of ICCAD,* pages 671-675, 1997.

[14] A. Mishchenko, S. Chatterjee, and R. Brayton, "DAG-aware AIG rewriting: A fresh look at combinational logic synthesis," In *Proc. DAC '06,* pp. 532-536, 2006

[15] M. Pedram and N. Bhat. "Layout-driven logic restructuring and decomposition," In *Proc. of ICCAD,* pages 134-137, 1991.

[16] M. Pedram and N. Bhat. "Layout driven technology mapping," In *Proc. of Design Automation Conf.,* pages 99-105, 1991.

[17] M. Yannakakis. "A polynomial algorithm for the min-cut linear arrangement of trees," In *Journal of the ACM,* Vol. 32, No. 4, pages 950-988, 1985.

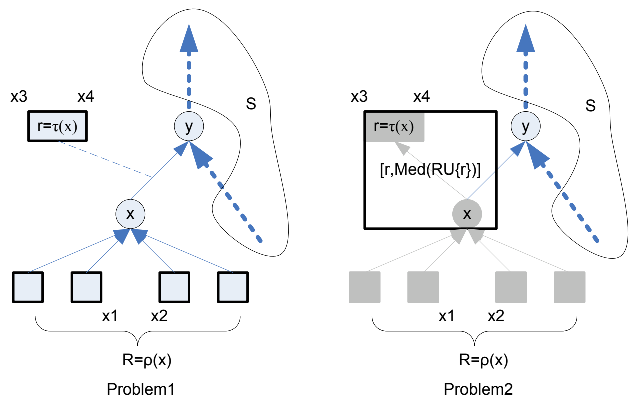

**Figure 4: Two node placement problems considered in the proof of optimality.**

## Appendix: Proof of Optimality

We prove the optimality of the DP algorithm only along the $x$-axis, since the objective function is separable. We assume that nodes are associated with one dimensional segments, instead of two dimensional rectangles.

The following notation will be used:

$$\begin{array}{lcll} R & = & \rho(x) & \\ \text{med}(R) & = & [x_1, x_2] & x_1 < x_2 \\ r & = & \tau(x) = [x_3, x_4] & x_3 < x_4 \\ \phi_R(x) & = & \sum_{r_i \in R} \text{cost}([r_i, x]) - \text{cost}(r_i) & \\ \text{cost}(r_i) & = & |r_i| & \end{array}$$

$$\begin{array}{ll} r_1 \prec r_2 & \text{if and only if }, \forall x \in r_1, \forall y \in r_2, x < y \\ r_1 \succ r_2 & \text{if and only if }, \forall x \in r_1, \forall y \in r_2, x > y \end{array}$$

Figure 4 describes two closely related node placement problems. In Problem 1, the fanins of nodes $x$ are associated with segments $R = \rho(x)$. Node $x$ also fans out to node $y$ as well as a fixed net $r = \tau(x)$. The rest of the nets are labeled $S$. In Problem 2, the fanin $x$ of node $y$ is associated with a segment $[r, \text{med}(R \cup \{r\})]$. The rest of the nets are the same $S$. We can explicitly write down the objective functions for the two problems. Here, we abuse notation and use $x$ to denote the position of node $x$.

$$\begin{aligned} f_1(x,y) &= \phi_R(x) + \mathrm{cost}([x,[y,r]]) + \mathrm{cost}(S) \\ f_2(y) &= \mathrm{cost}([y,[r,\mathrm{med}(R \cup \{r\})]]) + \mathrm{cost}(S) \end{aligned}$$

where $\mathrm{cost}(S)$ is the sum of the bounding box sizes of nets in $S$. Let

$$\begin{aligned} g(x,y) &= f_1(x,y) - f_2(y) \\ &= \phi_R(x) + \mathrm{cost}([x,[y,r]]) - \mathrm{cost}([y,[r,\mathrm{med}(R \cup \{r\})]]) \end{aligned}$$

**Lemma** 1.

$$[r, \mathrm{med}(R \cup \{r\})] = \begin{cases} r & \textit{if } r \cap \mathrm{med}(R) \neq \emptyset \\ [x_3, x_1] & \textit{if } r \prec \mathrm{med}(R) \\ [x_2, x_4] & \textit{if } r \succ \mathrm{med}(R) \end{cases}$$

PROOF. Let $R' = R \cup \{r\}$. If $r \cap \mathrm{med}(R) \neq \emptyset$, $\forall p \in r \cap \mathrm{med}(R)$, $|\mathcal{N}_l(p,R)| = |\mathcal{N}_r(p,R)|$, $x_3 < p < x_4$. Therefore, $|\mathcal{N}_l(p,R')| = |\mathcal{N}_r(p,R')|$. According to Theorem 1, $p \in \mathit{med}(R')$, or, $r \cap \mathrm{med}(R) \subseteq \mathrm{med}(R')$.

Let $r \cap \mathrm{med}(R) = [x_5, x_6]$. $x_5$ and $x_6$ are also in the end point set induced from $R'$. $\forall q, q < x_5$,

$$\begin{aligned} |\mathcal{N}_l(q,R')| &\leq |\mathcal{N}_l(p,R')| - 1 \\ |\mathcal{N}_r(q,R')| &\geq |\mathcal{N}_r(p,R')| + 1 \end{aligned}$$

Thus, $|\mathcal{N}_l(q,R')| < |\mathcal{N}_r(q,R')|$ and $q$ is not in $\mathrm{med}(R')$. Similarly, we can show that $\forall q > x_6$, $q$ is not in $\mathrm{med}(R')$. Therefore, $\mathrm{med}(R') = r \cap \mathrm{med}(R) \subseteq r$ and $[r, \mathrm{med}(R')] = r$.

If $r \prec \mathrm{med}(R)$, two end points $x_3$ and $x_4$ are added to the end point set of $R$. But $x_3$ and $x_4$ are both less than $\mathrm{med}(R)$. $x_1$ becomes the end point of $\mathrm{med}(R \cup \{r\})$. The other end point of $\mathrm{med}(R \cup \{r\})$ is between $x_1$ and $x_3$. Therefore, $[r, \mathrm{med}(R \cup \{r\})] = [x_3, x_1]$. □

**Lemma** 2. *Given segment $c$ such that $r \subseteq c$, the following 3 statements hold.*

1. *If $r \cap \mathrm{med}(R) \neq \emptyset$, then $\big(r \cap \mathrm{med}(R)\big) \subseteq \mathrm{med}(R \cup \{c\})$*
2. *If $r \prec \mathrm{med}(R)$, then $x_1 \in \mathrm{med}(R \cup \{c\})$*
3. *If $r \succ \mathrm{med}(R)$, then $x_2 \in \mathrm{med}(R \cup \{c\})$*

PROOF. If $r \cap \mathrm{med}(R) \neq \emptyset$, then $c \cap \mathrm{med}(R) \neq \emptyset$. Thus $\mathrm{med}(R \cup \{c\}) = c \cap \mathrm{med}(R)$ and hence $r \cap \mathrm{med}(R) \subseteq \mathrm{med}(R \cup \{c\})$.

If $c = [x_3, x_4], x_3 \leq x_4$, and $r \prec \mathrm{med}(R)$, then $c$ adds two points are added to the end point set of $R$. $x_3$ must be at the left side. After $x_3$ is added to $R$, there is an odd number of end points $R \cup \{x_3\}$. $x_1$ becomes the median of $R \cup \{x_3\}$. When $x_4$ is added, $x_1$ will still be included as one of minimizer of $R \cup \{c\}$. The third statement is proved similarly. □

The next three lemmas deal with the three cases in Lemma 2 and show lower bounds for $g(x,y)$ .

**Lemma** 3. *If $r \cap \mathrm{med}(R) \neq \emptyset$, then $g(x,y) \geq g(x_0,y) = \phi_R(x_0)$ where $x_0$ is any $x_0 \in r \cap \mathrm{med}(R)$.*

PROOF. By Lemma 1,

$$g(x,y) = \phi_R(x) + \mathrm{cost}([x,[y,r]]) - \mathrm{cost}([y,r])$$

Since $r \subseteq [y,r]$, by the first statement in Lemma 2, any $x_0 \in r \cap \mathrm{med}(R)$ is a minimizer of $\phi_{R \cup \{[y,r]\}}(x)$, which is $\phi_R(x) + \mathrm{cost}([x,[y,r]])$ by definition. Therefore,

$$\begin{aligned} g(x,y) =& \phi_R(x) + \mathrm{cost}([x,[y,r]]) - \mathrm{cost}([y,r]) \\ \geq& \phi_R(x_0) + \mathrm{cost}([x_0,[y,r]]) - \mathrm{cost}([y,r]) = g(x_0,y) \end{aligned}$$

and

$$\begin{aligned} g(x_0,y) =& \phi_R(x_0) + \mathrm{cost}([y,r]) - \mathrm{cost}([y,r]) \\ =& \phi_R(x_0) \end{aligned}$$

□

**Lemma** 4. *If $r \prec \mathrm{med}(R)$, then $g(x,y) \geq g(x_1,y) = \phi_R(x_1)$.*

PROOF. Since $r \prec \mathit{med}(R)$, from Lemma 1,

$$g(x,y) = \phi_R(x) + \mathrm{cost}([x,[y,r]]) - \mathrm{cost}([y,[x_3,x_1]])$$

Since, $r \subseteq [y,r]$, the second statement in Lemma 2 implies that $x_1$ is a minimizer of $\phi_{R \cup [y,r]}(x)$, which is $\phi_R(x) + \mathrm{cost}([x,[y,r]])$ by definition. Therefore,

$$\begin{aligned} g(x,y) =& \phi_R(x) + \mathrm{cost}([x,[y,r]]) - \mathrm{cost}([y,[x_3,x_1]]) \\ \geq& \phi_R(x_1) + \mathrm{cost}([x_1,[y,r]]) - \mathrm{cost}([y,[x_3,x_1]]) = g(x_1,y) \end{aligned}$$

and

$$\begin{aligned} g(x_1,y) =& \phi_R(x_1) + \mathrm{cost}([x_1,[y,x_3,x_4]]) - \mathrm{cost}([y,[x_3,x_1]]) \\ =& \phi_R(x_1) + \mathrm{cost}([y,x_1,x_3]) - \mathrm{cost}([y,x_3,x_1]) \\ =& \phi_R(x_1) \end{aligned}$$

□

**Lemma** 5. *If $r \succ \mathrm{med}(R)$, then $g(x,y) \geq g(x_2,y) = \phi_R(x_2)$.*

The proof is similar to that of Lemma 4.

**Theorem** 2.

$$\min_{x,y} f_1(x,y) = \phi_R(\tilde{x}) + \min_y f_2(y).$$

PROOF. By Lemmas 3-5, for each possible location of $\tau(x) = r$ with respect to $\mathrm{med}(R)$, we can find a point $\tilde{x}$ such that $g(x,y) \geq \phi_R(\tilde{x})$. Thus

$$\begin{aligned} \min_{x,y} f_1(x,y) &= \min_{x,y} \big(g(x,y) + f_2(y)\big) \\ &\geq \phi_R(\tilde{x}) + \min_y f_2(y) \end{aligned}$$

On the other hand,

$$\begin{aligned} \min_{x,y} f_1(x,y) &= \min_{x,y} \big(g(x,y) + f_2(y)\big) \\ &\leq g(\tilde{x},y) + f_2(y). \end{aligned}$$

Minimizing with respect to $y$, yields

$$\min_{x,y} f_1(x,y) \leq \min_y \big(g(\tilde{x},y) + f_2(y)\big) = \phi_R(\tilde{x})) + \min_y f_2(y)$$

Therefore,

$$\min_{x,y} f_1(x,y) = \phi_R(\tilde{x}) + \min_y f_2(y).$$

□

**Corollary** 1. *The DP algorithm is optimal.*

PROOF. Theorem 2 implies that Problem 1 in Figure 4 can be reduced to the smaller Problem 2. The algorithm follows this procedure. At each step in the forward pass, it constructs a segment and passes it on to the smaller problem with one less node. During the backward pass, the optimal solution $y$ of the smaller Problem 2 is chosen in the median of all the segments, including the bottom up segment and its fanout segment. Since the optimum cost of Problem 1 differs by a number independent of $y$, $\phi_R(\tilde{x})$, then $y$ is optimum for Problem 1. □