

Performance Driven Optimization of Network Length in Physical Placement

Wilm Donath Prabhakar Kudva
IBM T.J. Watson Research Center
Yorktown Heights, NY

Lakshmi Reddy
IBM Server Group
Hopewell Junction, NY

Abstract

The locations of individual circuits in a placement have a significant impact on the wire length and therefore on the overall timing of designs. A novel technique that moves sets of circuits (gates) during or after timing driven placement to improve performance of designs is proposed. An efficient method to identify optimal set of circuit movements to reduce wire length, called *strong motions* is presented. Experimental results with a min-cut placement tool indicate that the proposed approach of direct manipulation of circuit locations, improves the timing of partitions of a chip significantly.

1 Introduction

Advances in technology have placed new emphasis on design automation techniques that can better control interconnect (wire) lengths. Physical placement is an important step in determining of the length of the wires in a design. Timing driven placement [5, 15, 1] techniques used in typical chip design methodologies [9] incorporate constraints (such as net weights, capacitance/delay budgeting etc.) into a placement algorithm to improve the circuit (gate) locations for timing. The constraints used by these techniques are obtained from the timing information available at synthesis sign-off. These constraints may not be accurate due to the inability of most stand alone synthesis systems to predict wire capacitances. The timing driven placement algorithms therefore are unable to consistently optimize the critical regions resulting in placement that may often be sub-optimal for timing.

It is important for placement algorithms to work with an integrated timing analysis environment, to ensure that they work on the true critical regions and not necessarily the ones predicted by synthesis as critical. Also, very often, there is room for further improvement in timing by directly manipulating the locations of the circuits. In other words, circuit locations can be changed in a given placement in conjunction with incremental timing analysis [7] for timing optimization. This can be done either after an initial placement or during intermediate steps of placement.

The problem addressed in this paper is as follows: For a given placement of a design, which may either be the end result of a placement algorithm or an intermediate result (as in the case of a bipartitioning based placer), one would like to identify circuits (gates) in the design that can be moved to different locations to improve the overall timing of the design without violating the placement constraints. The problem of circuit movement involves identification of circuits to be moved as well as the selection of locations to which they need to be moved.

The problem of identification of the circuits to be moved is addressed as follows. Timing optimization is most effective if the circuits selected for movement are from the critical region of the design. The critical region may be identified by an incremental timing analysis engine [7] that uses existing placement locations in its timing calculations [8, 10]. Incremental timing analysis is performed on the physical netlist to confirm every timing gain

after applying a selected circuit move, thereby keep track of the critical region in an incremental fashion. Any suitable delay model [13, 2] which can be supported within a standard incremental timing analyzer may be used.

The main contribution of this paper is a method to select and apply optimal moves of circuits in a given critical region to optimize timing by reducing net lengths. Typically, it may be necessary to move a group of circuits. Many situations occur where individual circuit movements have no effect or could even worsen the timing of the circuit. On the other hand the collective motion of several circuits together may have significant improvements.

Consider the meander in a critical path in Figure 1. Moving only one of the circuits C , D or E would have no beneficial effect on the total net length. The movement of C , D and E together would reduce total net length and therefore improve timing. The problem becomes more complex if C , D or E have multiple fanins and fanouts. The proposed technique also addresses this issue. Similarly, consider a single net with three nodes connected using a steiner tree as shown in Figure 2. If we move any individual node A or B in the vertical direction, there is no reduction in net length (assuming orthogonal routing). If we move the two nodes together as shown the total net length is decreased. These simple examples illustrate the benefit of circuit motions and the need for a formalism to identify optimal motions that improve the *timing* of the design.

In this paper, we present location optimization techniques that exploit the concepts illustrated in these examples. The proposed techniques are applied in a selective manner so as to cause minimal perturbation to the existing placement. The techniques can also be used in conjunction with other standard synthesis and physical design optimizations [10, 17, 2, 6, 11].

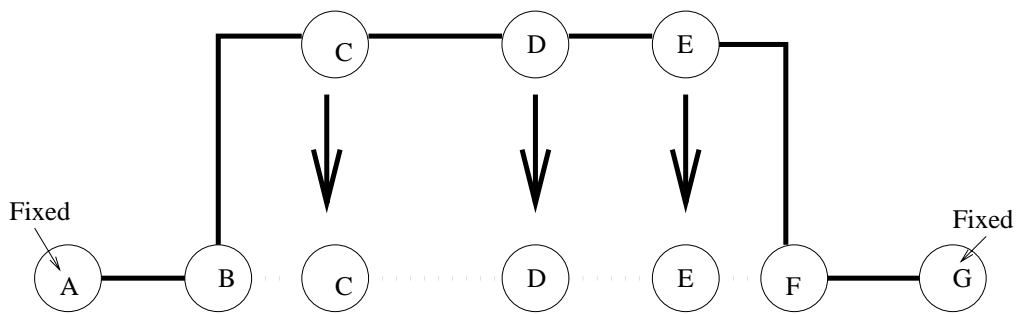


Figure 1: Changes in Locations for Critical Paths

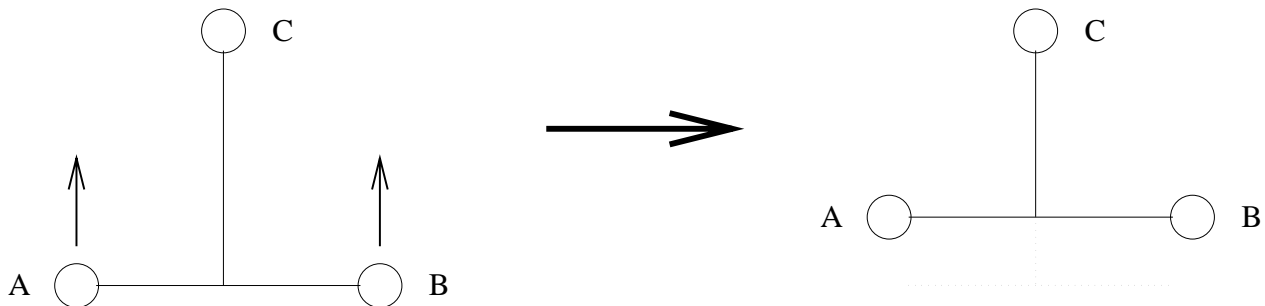


Figure 2: Motion of individual Circuits

The rest of the paper is organized as follows. Section 2 briefly discusses the prior art in this area. In Section 3, we discuss the representation and abstraction of the chip placement image which will be used in the rest of the paper. Section 4 will present the methodology scenarios during which the move selection algorithm can be applied. Section 5 develops a theory for the selection of optimal moves. Timing optimization algorithms based on the theory are developed in Section 6. Results are presented in Section 7 followed by conclusions.

2 Related Work

Earlier works have often used the ability to specify constraints into the placement algorithm such as net weights and capacitance targets to achieve similar goals [5, 15]. In our paper, we consider location modification algorithms for timing optimization that are tightly integrated with timing analysis. In [16], locations are specified as variables for timing improvement and an exact non-linear optimization problem is formulated to achieve this goal. The run time of non-linear methods tends to grow quickly with the size of the designs. The technique provided in our paper is intended to deal with fast location optimizations of large designs.

Recent wire planning techniques [6] focus on optimizing wire delay while ignoring the effect of gate delays during logic synthesis. Their approach gives a method to perform better logic synthesis optimizations on gates so as to improve the global wiring. The approach presented in this paper can be used in conjunction with most logic synthesis/timing driven placement techniques.

Our paper does not deal with optimal topology generation for routing of nets given fixed locations of circuits [2]. Instead, our paper focuses on selecting placement moves for the circuits themselves and leaves the generation of efficient topologies for the nets to techniques described in [2]. Also, the work described in our paper does not deal with the issues of interconnect sizing such as the methods described in [2] which are typically performed on actual wires during or after routing. The routed result of a placement derived after applying our method can be further subjected to interconnect optimization techniques. In this paper, the optimizations will be based on estimates of routed wire lengths that will be expected to be matched by a router. This is typically valid for chip methodologies in the absence of significant congestion in the design.

3 Placement Image

The placement of a design can be viewed at two levels of abstraction. Locations can either be defined as exact cartesian co-ordinates of the legal location of a given cell (called *detailed location*), or an abstract and approximate representation of the location (called *coarse location*). In the case of detailed locations, the circuits have precise legal locations for a given chip image and the circuit rows and wiring tracks are precisely defined. Alternately, one could divide the chip/design area into *bins* as shown in figure 3, where only abstracted information is maintained with respect to each bin. For example, each bin will have associated with it a certain cell capacity, wiring capacity, etc. The coarse image is especially beneficial in a synthesis/physical design environment where significant changes are made to the design and maintaining legal locations for detailed placement is too costly to compute. The discussions and algorithms presented in this paper, support both coarse and detailed views of the placement.

There are benefits to using the coarse image. In a synthesis/physical design integrated environment, the topology of the netlist could still be changing. Therefore the physical design data is abstracted enough to provide reasonable accuracy, while at the same time not burdening the transforms with details of physical data. Circuits can be moved between the bins without a complex legalization procedure. Instead, we keep track of a simpler measure of how much of the *bin capacity* is used up by circuits already placed in the bin. A significant amount of work is required to ensure wirability of the design in the following phases of detailed placement and routing. Coarse placement have functions that relate to the physical characteristics of the chip image: where and how many circuit locations are available, where i/o's are placed, where big partitions are placed and block space for other circuits, where power lines are placed and how they block other wiring. These data can be abstracted from a partial design and used by the algorithms. Within the bins, approximate wire lengths for all gates in the bins could also be computed using Rent rule [3, 4]. It should be noted that timing analysis can be done in such a design with an accuracy that is better than that one could do in the absence of any physical data.

The bins may be either generated on a full chip image given the number of regions and the accuracy required or the regions may be defined by a bipartitioning based placer in a scenario where the optimizations are performed

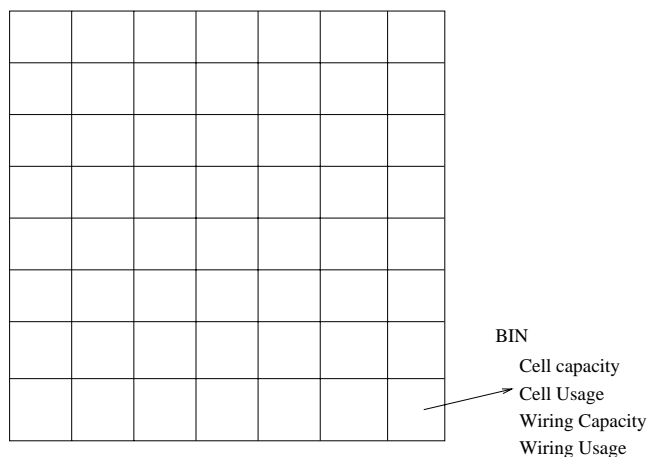


Figure 3: Coarse Image View

during the *cuts* of such a placer.

4 Optimization Scenarios

The circuit move algorithm for timing optimization presented in this paper may be applied in the following two scenarios as shown in figure 4. The first scenario is the application of the circuit move during placement. In this scenario, the circuit move algorithm is applied in concert with placement using a min-cut bipartitioning placer. The placer we use [8] works as follows: the placer performs cuts and each cut doubles the number of bins present, by making cuts alternately in the horizontal (x) and vertical (y) directions. After each cut the location of the cell has been narrowed down improving the ability to predict the wire lengths after that cut. This process is continued until the leaf cell is reached. The circuit move algorithm is applied after each cut of the placer. As the cuts progress, the algorithm applies more and more precise moves until a final placed design is reached. In the second scenario, the placement has already been performed and the designer would like to maximize the timing improvement with minimal change in placement locations. The algorithm is then applied on the placed design. The final step in the second scenario is the legalization step that legalizes the circuits perturbed by the moves applied by our algorithm.

The algorithm for the motion of circuits is called with a threshold for the improvement in slack required. A critical region is selected. The algorithm selects moves which may help in improving the critical region. These are called *Strong motions* and are described in section 5. Motions that do not interfere with other critical paths are selected and applied. A new critical path is found and the process is repeated until the improvement is less than the given threshold.

5 Theory of moves of circuits in a Manhattan Metric

In this section the theory for the sensitivity of a network to physical placement locations is developed. Methods to choose appropriate moves known as *strong moves* both for a single net and a set of nets are discussed. The goal is to select moves that maximize the timing improvement, although this method could be applied to any other metric such as wirability. As illustrated in Figures 1 and 2, several nodes may need to be moved at once. The question of how such sets are restricted to manageable and significant selections is considered. We require that moving a set of nodes should have an effect above and beyond the sum of the effects of any possible subsets. The

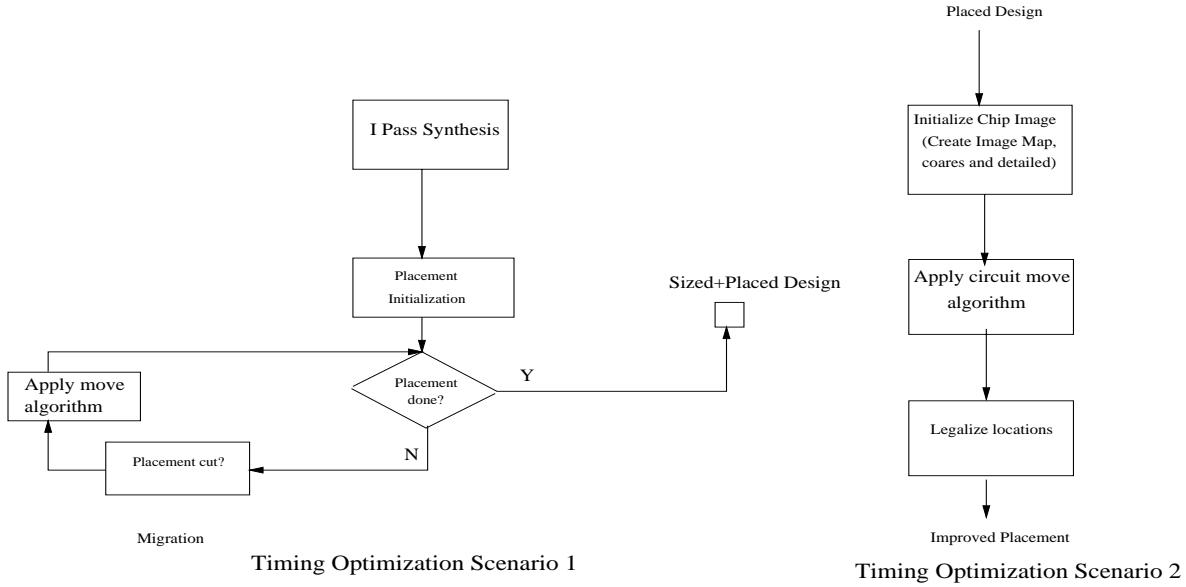


Figure 4: Scenarios for Circuit Move Optimizations

effect is given as the *sensitivity* of wire length with respect to the motion - where capacitance is proportional to the length of the wire required to connect up a net. Thus, in our definition of *strong motion* we give this concept mathematical rigor.

5.1 Stability of Motions

We shall now give some definitions, which allows us to define the concept of stable motions. A net n consists of nodes p_1, p_2, \dots, p_k , where each node p_i is at location x_i, y_i .

Definition 1 The configuration C_n of a net with k nodes is a vector $[(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)]$.

Definition 2 The motion $M(n)$ for a net n with configuration C_n is a vector $[(mx_1, my_1), (mx_2, my_2), \dots, (mx_k, my_k)]$, where (mx_i, my_i) can be one of $\{(-1,0), (1,0), (0,-1), (0,1), (0,0)\}$.

We will refer $M(n)$ as simply M in the following. As given in Definition 2 (mx_i, my_i) is one of five elemental motions. $(1,0)$ implies motion in the x direction and is denoted by $+x$. Likewise, $(-1,0)$ implies motion in the $-x$ direction and is denoted by $-x$. Similarly, $(0,1)$ and $(0,-1)$ are denoted by $+y$ and $-y$. $(0,0)$ implies no motion in any direction. Note that a node can only have an elemental motion in the x or y direction but not both.

The application of a motion M with magnitude u on C_n will result in a new configuration vector for net n , $C'_n = C_n + Mu = [(x_1 + mx_1 \cdot u, y_1 + my_1 \cdot u), (x_2 + mx_2 \cdot u, y_2 + my_2 \cdot u), \dots, (x_k + mx_k \cdot u, y_k + my_k \cdot u)]$.

Definition 3 A primitive motion is one where just one node moves in the $+x$, $-x$, $+y$, or $-y$ direction.

Example:

Let net X be connected to nodes A, B, C with $C_X = [(x_1, y_1), (x_1, y_2), (x_3, y_3)]$. Let $M_X = [(1,0), (0,0), (0,0)]$. Applying motion M_X to net X will move node A in the $+x$ direction. We will use the shorthand notation of $A < +x >$ to denote the movement of node A in the $+x$ direction. Note that M_X is a primitive motion.

The length $L(C_n)$ of wire required to interconnect the configuration C_n can be defined in several different ways. We are interested in Manhattan lengths. The length of wire is estimated to be that of the sum of the horizontal and vertical extent of the set of nodes. There are numerous techniques to estimate wire lengths for a set of points.

1. Often, for the sake of simplicity of computation, the minimal spanning rectangle is used as that measure $L(C_n)$: it is trivial to find cases for nets with 4 or more nodes where that length is inaccurate. [14]
2. The minimal spanning tree length can be used for $L(C_n)$: it is of course a pessimistic measure in many cases [14].
3. The Steiner tree length is a commonly used measure. One is allowed to generate intersection nodes, where three or four wires connect, so that it gives the shortest length of wire required for interconnecting the set of nodes in a rectilinear system [14] [12]

A fundamental theorem for constructing minimal length Steiner trees is

Theorem 1 (Hanan [12]) *Consider the set of nodes that are needed to complete any rectangle formed by any pair of nodes p_i and p_j of the rectangle. For a Steiner tree to have minimal length, the intersection nodes can be any subset of the original nodes of the net plus the rectangle-completing set of nodes.*

Example:

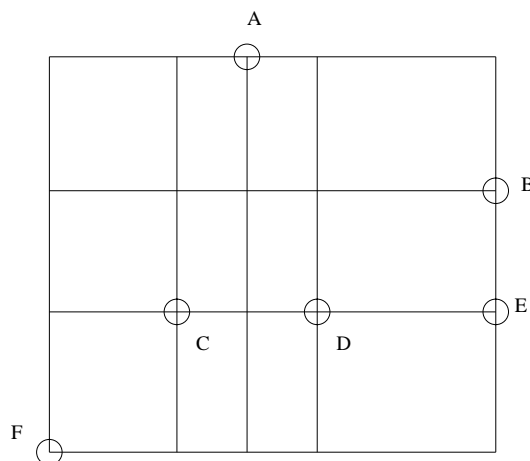


Figure 5: Example of a Hanan Grid

The example in figure 5 shows six nodes labeled A, B, C, D, E, F and the associated Hanan grid. Note that nodes $C, D,$ and E are aligned horizontally, and nodes B and E are aligned vertically.

Example:

In figure 6 the nodes are shown: node A is moving in the $+x$ direction, nodes B and E are moving in the $-x$ direction, nodes C and D are moving in the $-y$ direction, and F in the $+y$ direction.

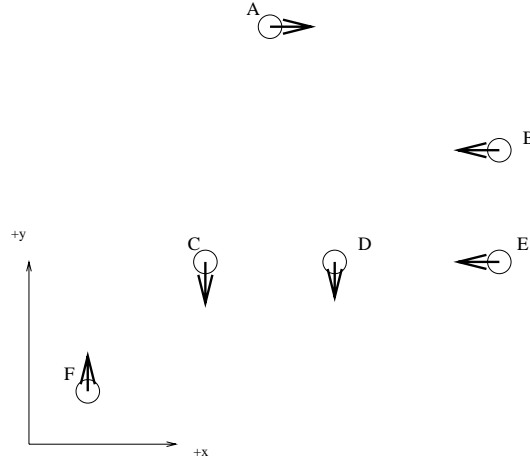


Figure 6: Example of some motions of nodes in a net

Definition 4 Sensitivity of $L(C_n)$ for a given motion M , denoted by $\frac{dL(C_n)}{dM}$, is defined as follows:

$$\frac{dL(C_n)}{dM} = \lim_{u \rightarrow 0} \frac{L(C_n + M \cdot u) - L(C_n)}{u} \quad (1)$$

where u is some scalar.

In other words, $\frac{dL(C_n)}{dM}$ gives the change in wire length for net n , due to a change in C_n (the locations of points in a net), for a given motion M of infinitesimal magnitude u .

Example:

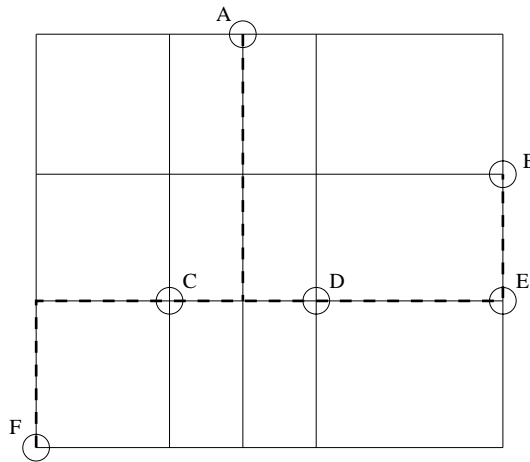


Figure 7: Example of net with Steiner Interconnections

Figure 7 shows the six nodes of figure 5 with interconnections denoted by the broken lines. The length of these interconnections would not be changed if either nodes B or E moved incrementally to the left. So for either of these two motions, $\frac{dL(C_n)}{dM}$ would be 0. The joint motion of B and E to the left would result in $\frac{dL(C_n)}{dM} = -1$. Moving A downward ($-y$) would also give a sensitivity of -1 . Moving A either left or right (i.e. $-x$ or $+x$)

would not change the total length. Moving A up (+y) would give a sensitivity of 1. Moving F either up or right (i.e. +x or +y) would give a sensitivity of -1.

We define the notion of *composition* of two vectors M_1 and M_2 denoted $M_1 + M_2$

Definition 5 Two motions $P = [(px_1, py_1), (px_2, py_2), \dots, (px_k, py_k)]$ and $Q = [(qx_1, qy_1), (qx_2, qy_2), \dots, (qx_k, qy_k)]$ for some configuration C_n are composable if for $1 \leq i \leq k$, either $(px_i, py_i) = (0, 0)$ or $(qx_i, qy_i) = (0, 0)$. The composed motion M will be denoted by $P + Q = [(px_1 + qx_1, py_1 + qy_1), (px_2 + qx_2, py_2 + qy_2), \dots, (px_k + qx_k, py_k + qy_k)]$

Definition 6 A motion M_c is a component of M if all vector elements in M_c either have the same value as the corresponding vector element in M or are $(0, 0)$.

Definition 7 A motion M_c is a proper component of M if it is a component of M and is not identical to M and has a non-zero vector element.

Definition 8 Two primitive motions M_1 and M_2 of nodes p_1 and p_2 are parallel if the two nodes have the same x coordinate and go in the same +y direction (or -y direction) or if the two nodes have the same y coordinate and go in the same +x or -x direction.

Definition 9 A motion M is called parallel if the nodes associated with M that have non-zero move components are either aligned at the same y -coordinate and the motion is either in the +x or -x direction or are aligned at the same x -coordinate and the motion is either in the +y or -y direction. In other words, the motion is perpendicular to the line of the alignment.

Often there are several implementations of the Steiner tree or the minimal spanning, which all have the same minimal length. If we apply a given motion, and attempt to retain the implementation, then one or the other implementation could be favored. $\frac{dL(C_n)}{dM}$ is determined by the implementation that gives the smallest sensitivity.

Finally, we define the concept of *stability*:

Definition 10 A motion M is stable if

- M is a primitive motion or
- for any decomposition of M ,

$$M = M_1 + M_2,$$

where M_1 and M_2 are non-zero,

$$\frac{dL(C_n)}{dM} < \frac{dL(C_n)}{dM_1} + \frac{dL(C_n)}{dM_2}$$

Example:

In illustrating the above definition, we use a simple example shown in Figure 8, which has two equivalent implementations. We note that the distances $A - B$ and $C - D$ are equal and that the distance $B - C$ is larger than the distance $A - B$. We see that in one of the implementations B is connected with C , and in the other A is connected with D . The following sensitivities can be established.

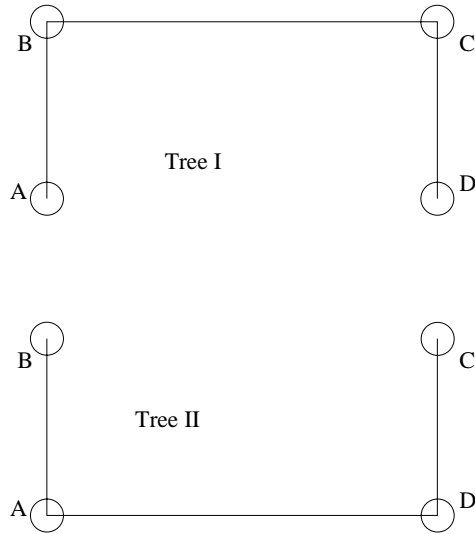


Figure 8: Two implementations of a simple tree

Motion	Sensitivity		
	Tree I	Tree II	Best
$A < +x >$	0	0	0
$A < -y >$	1	1	1
$A < +y >$	-1	0	-1
$A < -x >$	1	1	1
$B < +x >$	0	0	0
$B < -y >$	0	-1	-1
$B < +y >$	1	1	1
$B < -x >$	1	1	1
$A < +x > B < +x >$	-1	-1	-1
$A < -x > B < -x >$	1	1	1
$A < +y > D < +y >$	-2	-2	-2

Symmetry gives us the results for the motions of C and D . We can see that the motions $(A < -x >, B < -x >)$ and $(A < +x >, B < +x >)$ are stable. On the other hand, the motion $(A < +y >, D < +y >)$ is **not** stable.

Example:

We refer back to figure 7 and remember that for moving either nodes B or E to the left, the sensitivity $\frac{dL(C_n)}{dM}$ is 0. However, the joint motion of B and E to the left yields a sensitivity of -1 , and therefore that motion is *stable*. Also note that the upward motion of either D or E yields a sensitivity of 0, while their joint upward motion yields a sensitivity of -1 . The upward motion of node C yields a sensitivity of 0: the joint upward motion of C , D , and E yields a sensitivity of -1 , which is **not** less than the sum of the motion of C and the joint motion of D and E , and therefore is not a *stable* motion.

Theorem 2 *If one uses Manhattan metric, and the cost measure is based either on minimum spanning rectangle, or minimal spanning tree, then only parallel motions can be stable (or if a motion is stable it is also parallel).*

Proof: The proof is given in the appendix.

Conjecture 3 *Theorem 2 holds also when the Steiner Tree is used as the cost measure.*

This conjecture is a reasonable heuristic even with Steiner tree estimates. We believe that any useful moves in violation of Conjecture 3 would be rare.
not be worthwhile.

5.2 Stability of Nets

In this section, we extend motions to a set of nets.

Definition 11 *Consider a set S of nets n_1, n_2, \dots, n_m . Assume there exists a weight vector w_1, w_2, \dots, w_m corresponding to the nets, where each w_i is greater than 0. Let*

$$L(S) = \sum_{i=1}^m w_i L(C_{n_i}) \quad (2)$$

be the length of set S .

We can find all nodes p_1, p_2, \dots, p_k which belong to at least one net in S , and construct the configuration for S , $C(S) = [(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)]$. Motions for S may also be constructed in a manner similar to that for a single net.

Lemma 1 *For any motion M*

$$\frac{dL(S)}{dM} = \sum_{i=1}^m w_i \frac{dL(C_{n_i})}{dM}$$

For the case of the weighted sum of length of nets we can also define *stable sets of motions* exactly as in definition 10. The properties embodied in Definition 10 hold for the function $L(S)$. The theorem 2 also extends to motions with sets of nets and the length $L(S)$:

Theorem 4 *A motion M cannot be a stable motion for $L(S)$ if it is not a parallel motion.*

Proof: This corollary is a direct conclusion from theorem 2.

5.3 Strong Motions

The basic interest of algorithms in this field is to find moves which have large negative sensitivities: we shall introduce therefore a stronger criterion than stability for motions.

Definition 12 *A strong motion M is a stable motion which has the property that for all proper components M_c of M*

$$\frac{dL(S)}{dM} < \frac{dL(S)}{dM_c} \quad (3)$$

Should M be primitive it can be considered as a strong motion.

To determine whether a motion M is *strong*, in the worst case one may have to test whether Definition 12 holds for all sub-components. However, should a proper component M_c of M be strong, then one need not test against any subcomponent of M_c .

Since for motion M to be strong, it has to be stable. Theorem 4 can therefore be extended to strong motions as follows.

Theorem 5 *For a motion M to be strong all primitive components of M must be parallel with each other.*

6 Circuit Move Algorithm

Let S be the set of nets in the critical region of a design. The move optimization algorithm computes the set of strong motions which give negative sensitivity (improve wire length) M_S for all the individual nets in the critical region as described in Section 6.1. An element of M_S may have negative or positive sensitivity for $L(S)$ even though it has a negative sensitivity for the individual net for which it was computed. The algorithm divides the M_S into a two subsets, L_{neg} and L_{pos} . L_{neg} (L_{pos}) has members of M_S that have negative (positive) sensitivities for $L(S)$. It is possible that some members of L_{pos} may be combined to form new strong moves which have negative sensitivities in $L(S)$. Such moves are also identified and added to L_{neg} . This process is further described in section 6.2. Finally, members of L_{neg} are applied to the design to improve the timing as described in Section 6.3

6.1 Computation of strong motions for a single net

There are several sets of motions which constitute the complete of *strong* motions for a single set:

1. the primitive (see definition 3) motions of the individual nodes
2. Should more than one node be at the same coordinate, one would also test the joint movement of all the nodes at that coordinate
3. Combined motions from 1 or more sets of positions, where the positions have either the same value of x or y and the motion is perpendicular to the alignment of the positions. Here we may note that the only time such a motion can be strong is when a contiguous subset (or the complete set) of the positions along the alignment moves in lockstep

Except for the first item - the primitive motions of the individual nodes - all other motions need testing whether their first sensitivity is less than any of its component motions.

6.2 Computation of motions for a set of nets

Our main interest is in finding motions that have a negative sensitivity for $L(S)$. We are particularly interested in finding motions with negative sensitivities for some $L(C_n)$, where n is some net. For this algorithm we use the result - Theorem 5.

- Compile a set M_S of *strong* motions for all individual nets in S .
- Set $L_{neg} =$ list of motions in M_S with negative sensitivities in $L(S)$ (Definition 11)
 - $C(M) = M_S - L_{neg}$
 - while $C(M) \neq \phi$
 - * Take a motion M_c from $C(M)$
 - * If $\frac{dL(S)}{dM_c}$ is negative in $L(S)$, enter it in L_{neg}
 - * else
 - $E(M_c) =$ the set of nets that are connected to nodes in M_c
 - for all $n \in E(M_c)$ where $\frac{dL(C_n)}{dM_c}$ is positive, do
 - if there is a *strong* motion M^n in n such that both $M^n \cap M_c$ and $M^n - M_c$ are not empty, then enter $M^n \cup M_c$ as a candidate in $C(M)$
- return L_{neg}

Table 1: Results for the Circuit Move Algorithm

Circuit	move?	Gate Count	Delay	Slack	% Delay Improv.	Horiz	Vert
Des1	no	1363	3.525	-1.225		245/160	242/190
	yes			-0.95	7.8	234/163	263/192
Des2	no	3168	2.525	-1.075		327/251	419/316
	yes			-0.825	9.8	335/249	445/327
Des3	no	1392	3.35	-1.075		201/148	297/202
	yes			-0.95	3.73	210/150	296/199
Des4	no	1029	3.025	-0.725		135/ 95	157/109
	yes			-0.425	10	140/ 96	141/107
Des5	no	2254	2.525	-0.75		246/177	340/230
	yes			-0.475	10.8	263/179	377/239

6.3 Application of Strong Motions

The algorithm for the motion of circuits is called with a threshold for the improvement in slack required. A critical path is selected, *strong motions* which are described in section 5 are identified that may help improve the path. The experiments were performed with scenario 1 or Figure 4, they can similarly be extended for scenario 2. The capacities of the bins are checked before application of the move and a move is applied only if there is enough room in the target bins. The gain in timing is verified by actual timing analysis on the selected moves and the any individual move is accepted only if it provides benefit a significant timing improvement. A new critical path is found and the process is repeated until the improvement is less than the given threshold.

7 Results

The experiments were run using timing optimization scenario 1 from figure 4. The placement tool used is part of a high performance design methodology [8]. Table 1 gives results for a number of test cases selected from timing aggressive designs. All timing information has been multiplied by an undisclosed number to protect actual timing information.

We see that in all test cases we have a significant improvement in timing - typically about 4 to 10% of the critical path. Wirability was measured in terms of the horizontal and vertical wires cut, and we give both the peak and the average wires cut. We can see in a few cases some degradation of wirability and which we plan to control by introducing wirability into our cost function. In almost all the test cases the circuit move algorithm moved less than 1 % or the circuits in the design. The placement algorithm was able to provide legal placement results after the moves were performed in all the cases.

Overall we observe a significant improvement in highly timing constrained designs. These designs had already been aggressively optimized for timing. The impact on wirability appears to be moderate. The experiments were run on small sized partitions of a chip and we expect the results to improve significantly with larger partitions due to the presence of longer wires.

8 Conclusions and Future Work

In this paper we presented the theory and efficient algorithms to improve the performance of a design by the movement of sets of circuits during or after timing driven placement. Our results indicate a significant improvement in the timing of designs and the impact on wirability appears to be moderate. The methods described in this paper can be further extended to support more sophisticated moves that involve critical as well as non-critical regions and target a variety of metrics including wirability, noise, yield and manufacturability.

Acknowledgements

The authors would like to thank Charlie Bivona, Joachim Clabes, Lisa Lacey, Greg Northrop, Dan Ostapko, Leon Stok, Paul Villarrubia for continued support to the PDS project. Comments on this paper from Dan Brand, John Darringer, Reinaldo Bergamaschi are appreciated.

References

- [1] ALPERT, C., CHAN, T., A.B.KAHNG, MARKOV, I., AND MULET, P. Faster minimization of linear wirelength for global placement. *IEEE Transactions on Computer-Aided Design* 17, 1 (Jan. 1998).
- [2] CONG, J., HE, L., KOH, C.-K., AND MADDEN, P. H. Performance optimization of VLSI interconnect layout. *Integration* 21 (1996), 1–94.
- [3] DONATH, W. E. Equivalence of memory to random logic. *IBM Journal of Research and Development* (September 1974), 401–407.
- [4] DONATH, W. E. Wire length distribution for placements of computer logic. *IBM Journal of Research and Development* (May 1981), 152–155.
- [5] DONATH, W. E., NORMAN, R. J., AGRAWAL, B. K., S. E. BELLO, S. Y. H., KURTZBERG, J. M., LOWY, P., AND MCMILLAN, R. I. Timing driven placement using complete path delays. In *Proc. ACM/IEEE Design Automation Conference* (June 1990), IEEE Computer Society Press.
- [6] GOSTI, W., NARAYAN, A., BRAYTON, R. K., AND SANGIOVANNI-VINCENTELLI, A. L. Wireplanning in logic synthesis. *International Workshop in Logic Synthesis* (1998), 520–529.
- [7] HATHAWAY, D., ABATO, R., DRUMM, A., AND VAN GINNEKEN, L. Incremental timing analysis. Tech. rep., 1996. IBM, U.S. patent 5,508,937.
- [8] HOJAT, S., AND VILLARUBIA, P. An integrated placement and synthesis approach for timing closure of PowerPC microprocessors. *Proc. International Conf. Computer Design (ICCD)* (1997), 206–210.
- [9] K.L.SHEPARD, AND ET.AL. Design methodology for the S/390 parallel enterprise server g4 microprocessors. In *IBM Journal of Research and Development* (July/September 1997), pp. 515–547.
- [10] LEE, M. T.-C., AND ET. AL. Incremental timing optimization for physical design by interacting logic restructuring and layout. *International Workshop in Logic Synthesis* (1998), 508–513.
- [11] LOU, J., SALEK, A., AND PEDRAM, M. Exact solution to simultaneous technology mapping and linear placement problem. *Proc. International Conf. Computer-Aided Design (ICCAD)* (1997).
- [12] M.HANAN. On steiner’s problem with rectilinear distance. *J.SIAM Appl.Math* 14, 2 (Mar. 1966), 255–265.
- [13] PILEGGI, L. Timing metrics for physical design of deep submicron technologies. *Proc. International Symposium on Physical Design* (1998), 28–33.
- [14] PREAS, B. T., AND KARGER, P. G. Placement, assignment, and floorplanning. In *Physical Design Automation of VLSI Systems*, B. Preas and M. Lorenzetti, Eds. 1988, ch. 4, pp. 87–155.
- [15] SARRAFZADEH, M., KNOL, D., AND TELLEZ, G. Unification of budgeting and placement. *Proc. ACM/IEEE Design Automation Conference* (1997), 758–761.
- [16] SRINIVASAN, A., CHAUDHARY, K., AND E.S.KUH. RITUAL: A performance driven placement algorithm for small cell ICs. In *Proc. International Conf. Computer-Aided Design (ICCAD)* (Nov. 1991), pp. 48–51.
- [17] STENZ, G., AND ET.AL. Timing driven placement in interaction with netlist transformations. *Proc. International Symposium on Physical Design* (1997).

Appendix:Proofs

8.1 The Main theorem on stability

Theorem 2 *If one uses Manhattan metric, and the cost measure is based either on minimum spanning rectangle, or minimal spanning tree, then only parallel motions can be stable (or if a motion is stable it is also parallel).*

Proof: For the case of the spanning rectangle, the proof is relatively simple and can be omitted. For the case of the spanning tree, we have to develop some mathematical machinery; for one, we have the complication that there may be several optimum solutions. In this part we develop the necessary machinery.

We also mention here that from theorem 2 and the conjecture 3 we find that

Corollary 1 *A motion M cannot be an stable motion for $L(S)$ if it is not a parallel motion.*

8.1.1 Implementations and solutions

We first define the concept of a *implementation* and a *solution*:

Definition 13 *For the spanning tree problem, an implementation is a tree $I(C_n)$ with edges $[p_i, p_j]$ (where p_i and p_j are nodes of n such that there exists a path between any pair of nodes in n). The $\text{Length}(I(C_n))$ is just the sum of the distances of the edges in the implementation - i.e.*

$$\text{Length}(I(C_n)) = \sum_{[p_i, p_j] \in I} \ell(p_i, p_j)$$

It is to be noted that the functions $\ell(p_i, p_j)$ are continuous functions when the nodes p_i or p_j are displaced. Now for the definition of a solution:

Definition 14 *A solution $S(C_n)$ for either the minimal spanning tree problem or the Steiner problem is an implementation $I(C_n)$ such that*

$$\text{Length}(S(C_n)) \leq \text{Length}(I(C_n))$$

We denote by $\mathcal{S}(C_n)$ the set of all solutions for the set of nodes n with the locations given by C_n .

Let δ be an *infinitesimal* quantity - i.e so small, that it is much less than any separations of the nodes in n . Then

Lemma 2 *if there exists a solution $S(C_n + M\delta)$, then this is also a solution when $\delta = 0$ - i.e. $S(C_n + M\delta) \in \mathcal{S}(C_n)$ or, alternatively, $\mathcal{S}(C_n + M\delta) \subset \mathcal{S}(C_n)$*

Proof: $\text{Length}(S(C_n + M\delta))$ is a continuous function of δ , since all the functions $\ell([p_i, p_j])$ are continuous. Since $\text{Length}(S(C_n + M\delta)) \leq \text{Length}(I(C_n + M\delta))$, this means that $\text{Length}(S(C_n)) \leq \text{Length}(I(C_n))$. It is also clear that the reverse relationship does not hold.

The concept of *stability* of a motion has so far been defined only for the solution under either the minimum spanning rectangle, the minimal spanning tree, or the minimal Steiner Tree measure. We now define

Definition 15 *that a motion M is stable under an implementation $I(C_n)$ if there exists no decomposition of the motion into components M_1, M_2, \dots, M_k such that*

$$\frac{d\text{Length}(I(C_n + M\delta))}{d\delta} \geq \sum_{i=1}^k \frac{d\text{Length}(I(C_n + M_i\delta))}{d\delta}$$

To derive an important lemma

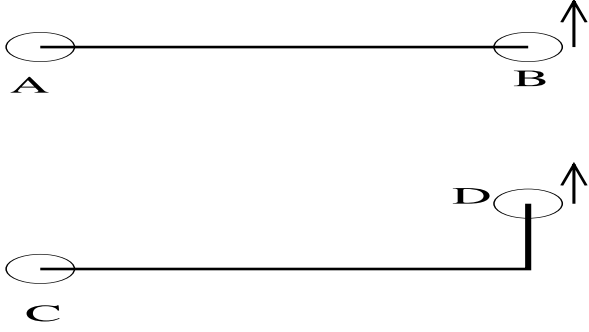


Figure 9: Different types of motions and their effect on lengths of connections

Lemma 3 *if a motion M is stable for a given graph n and position set C_n , then it is stable for every solution $S(C_n + M\delta)$.*

Proof: Let us denote by $S_g \in S(S_n)$, $S_i \in S(S_n + M_i\delta)$, and $S_M \in S(S_n + M\delta)$. Then we have

$$\frac{dLength(S_g(C_n + M_i\delta))}{d\delta} \geq \frac{dLength(S_i(C_n + M_i\delta))}{d\delta}$$

By hypothesis, M is a stable motion: therefore

$$\frac{dLength(S_M(C_n + M\delta))}{d\delta} < \sum_{i=1}^k \frac{dLength(S_i(C_n + M_i\delta))}{d\delta}$$

which, on account of the above inequality, becomes

$$\frac{dLength(S_M(C_n + M\delta))}{d\delta} < \sum_{i=1}^k \frac{dLength(S_M(C_n + M_i\delta))}{d\delta}$$

An obvious corollary to the lemma 3 is

Corollary 2 *If a motion is not stable for one solution, then it is not stable.*

8.1.2 Length Functions and motions

Consider figure 9: if a motion is perpendicular to a pair of aligned connected nodes, then the length of the connection increases proportional to the absolute size of the motion; on the other hand, if the motion would be parallel to the alignment, or if the two nodes would not be aligned, then the length of the connection increases or decreases by the size of the motion. In the first case, the sensitivity would not be continuous; in the second case, as long as the motion is *infinitesimal*, the sensitivity is continuous.

If we move a single node in its environment by an amount δx in the x-direction, then the lengths of its connections would change by an amount $a(\delta x) + b|\delta x|$; a similar formula exists for a motion δy in the y-direction as $c(\delta y) + d|\delta y|$. When several nodes move in a concerted way, i.e. supposing node A moves by δx then node B moves also by δx and node C moves in the y-direction by $-\delta x$ and node D moves also δx in the x-direction, then the total cost of the move still could be expressed by the same formula - i.e. $\Delta Length = a(\delta x) + b|\delta x|$; this is because for each node its separate contribution could be covered by the same formula (with different coefficients, of course) and the change of length of the connections could also be covered by the same formula.

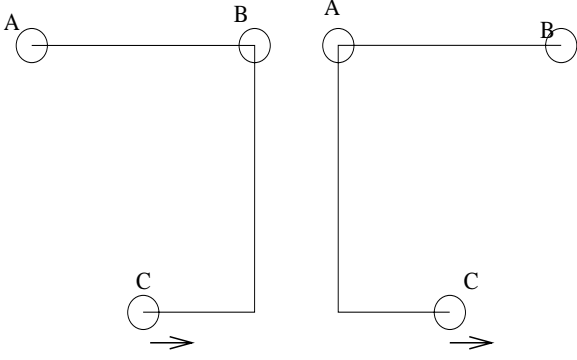


Figure 10: Three nodes connected perpendicularly with attached trees

8.1.3 Joint motions of non-aligned nodes cannot be stable

We now consider the situation where two nodes n_1 and n_2 are connected in an implementation, and that their difference in location is given by $\Delta x = x_2 - x_1$ and $\Delta y = y_2 - y_1$. If either of these is moved independently (with or without other nodes moving jointly with them on their side of the tree) the x-direction by δ_x then the cost is given as $a_i \delta x_i + b_i |\delta x_i|$, or if the amount is given by δy_i , then the cost is given as $c_i \delta y_i + d_i |\delta y_i|$. (note that any edge cuts a tree into two parts). The connection between the two nodes changes in length by $|\Delta x + \delta x_2 - \delta x_1| + |\Delta y + \delta y_2 - \delta y_1|$. Notice that in this case the sensitivities on the side trees simply add, and do not provide any contribution to any sensitivity wrt. to the joint motion of n_1 and n_2 . If either Δx or Δy is zero, then there is a contribution to the joint motion: if Δx is zero, then the motion of the two nodes in the y-direction with the same magnitude yields a sensitivity of 0, while either independently has a sensitivity of 1. We shall call a connection for which both Δx and Δy are non-zero a *non-aligned connection*. This argument gives us the result

Lemma 4 *Two nodes, which can only be connected by paths that include a non-aligned connection, cannot be included in a stable pair.*

We now wish to consider other possibilities how non-aligned nodes can be connected; they can either have a path through a non-aligned connection, in which case they can not be jointly in a stable set; fig 10 shows the other possibility, whereby two non-aligned nodes have a path between them without any non-aligned connections on the path. W.l.g we place the first node at $0, 0$, the second at $\Delta x, 0$ and the third at $\Delta x, \Delta y$. Each of the nodes may have an attached tree, and its cost for a motion in the x-direction is given as $a_i \delta x_i + b_i |\delta x_i|$ and in the y-direction by $c_i \delta y_i + d_i |\delta y_i|$. However, we note that for any combination of the three nodes moving jointly, these costs do not change the difference of the sensitivity of the joint motion as compared to the sum of the sensitivities of the individual motion. Therefore the only important terms are in the length of the connections $A - B$ and $A - C$. This function is given as

$$L(A - B) = \Delta x + \delta x_B - \delta x_A + |\delta y_B - \delta y_A|$$

$$L(B - C) = \Delta y + \delta y_C - \delta y_B + |\delta x_C - \delta x_B|$$

A thorough investigation of this formula will convince us that only *vertical motions* of A and B would be stable and *horizontal motions* of B and C would be stable, and joint motions of A and C are never stable.

This completes the proof of the theorem 2.