RC23695 (W0508-095) August 17, 2005
Electrical Engineering

IBM Research Report

Minimizing Wire Length in Floorplanning

Xiaoping Tang
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Ruiqi Tian
Freescale Semiconductor
Austin, TX 78721

Martin D. F. Wong
University of Illinois
Urbana, IL 61801

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Minimizing Wire Length in Floorplanning

Xiaoping Tang
IBM T.J. Watson Research
Yorktown Heights, NY 10598
U.S.A.
xtang@us.ibm.com

Ruiqi Tian
Freescale Semiconductor
Austin, TX 78721
U.S.A.
Ruiqi.Tian@freescale.com

Martin D.F. Wong
University of Illinois
Urbana, IL 61801
U.S.A.
mdfwong@uiuc.edu

Abstract

Existing floorplanning algorithms compact blocks to the left and bottom. Although the compaction obtains an optimal area, it may not be good to meet other objectives such as minimizing total wire length which is the first-order objective. It is not known in the literature how to place blocks to obtain an optimal wire length. In this paper, we first show that the problem can be formulated as linear programming. Thereafter, instead of using the general but slow linear programming, we propose an efficient min-cost flow based approach to solve it. Our approach guarantees to obtain the minimum total wire length in polynomial time and meanwhile keep the minimum area by distributing white space smarter for a given floorplan topology. We also show that the approach can be easily extended to handle constraints such as fixed-frame (fixed area), IO pins, preplaced blocks, boundary blocks, range placement, alignment and abutment, rectilinear blocks, soft blocks, one-dimensional cluster placement, and bounded net delay, without loss of optimality. Practically, the algorithm is so efficient in that it finishes in less than 0.4 seconds for all MCNC benchmarks of block placement. It is also very effective. Experimental results show we can even improve the wire length of very compact floorplans by 4.2%. Thus it provides an ideal way of post-floorplanning (refine floorplanning).

1. Introduction

Floorplanning is to decide the positions of circuit blocks or IP blocks on a chip subject to various objectives. It is the early stage of physical design and determines the overall chip performance. Due to the enormous complexity of VLSI design with continuous scaling-down of technology, a hierarchical approach is needed for the circuit design in order to reduce runtime and improve solution quality. Also, IP (module reuse) based design methodology becomes widely adopted. This trend makes floorplanning even more important.

Floorplanning has been studied for many years. Floorplan can be classified into two categories: slicing and non-slicing. Among slicing representations, there are binary tree[16] and normalized Polish expression[20]. For non-slicing structure, many representations have been invented recently, such as topology representation (BSG[15], sequence pair[14], TCG[12]), packing representation (O-tree[8], B*-tree[5]), and mosaic representation (CBL[9], Q-sequence[17], twin binary tree[23], twin binary sequence[24]). All of these algorithms compact blocks to the left and bottom, subject to the given topological constraints. Recently, additional constraints have been addressed in floorplanning, such as fixed frame[18, 1], alignment and performance (bounded net delay)[19], buffer planning in floorplanning[13], etc.. Again, within the approaches, the floorplan is compacted to lower-left (or upper-right) corner and then evaluated. In general, compaction implies minimum area. However, it may be sub-optimal for other objectives, such as minimizing wire length, routing congestion, and buffer allocation. As we can see, even with the same minimum area and the same topology, there exist lots of different floorplans that have different distribution of white space and thus have different values on other objectives. We illustrate the problem by a simple example in Figure 1.

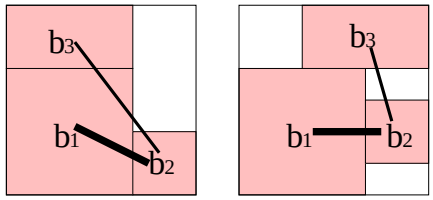


Figure 1: (a) The floorplan compacting blocks to the lower-left corner. However, the wire length is not optimal. (b) The floorplan with optimal wire length, which has the same topology and area but different distribution of white space. The dimensions for the 3 blocks are: $b_1(4\times4)$, $b_2(2\times2)$, $b_3(4\times2)$.

We observe that in floorplanning and placement, minimizing total wire length is the first-order objective. If a floorplanner/placer can minimize total wire length very well, then there is much freedom and space to consider and tradeoff other concerns such as routability and timing. Thus in the paper, we study the problem of minimizing total wire length in floorplanning, while leaving the consideration of routability and timing optimization (including buffer insertion) for future work.

It is not known in the literature how to place blocks to obtain an optimal wire length. We first show that the problem can be formulated as linear programming. Then, we find it can be solved by efficient min-cost flow implementation instead of general but slow linear programming. The approach guarantees to obtain the

minimum total wire length for a given floorplan topology. We also show that the approach is capable of handling various constraints such as fixed-frame (fixed area), IO pins, pre-placed blocks, boundary blocks, range placement, alignment and abutment, rectilinear blocks, soft blocks, one-dimensional cluster placement, and bounded net delay, without loss of optimality. It is an exact algorithm to minimize wire length and meanwhile keep the minimum area by distributing white space smarter, as well as to optimize the composite cost of both area and wire length. The algorithm is so efficient in that it finishes in less than 0.4 seconds for all MCNC benchmarks of block placement. It is also very effective. Experimental results show we can even improve the wire length of very compact floorplans by 4.2%. Thus it is worth applying as a step of post-floorplanning (refine floorplanning). It is noted that researchers have studied the problem of allocating white space in placement for various objectives[10, 22, 4, 2]. These methods are heuristics in terms of minimizing wire length. Our approach optimally minimizes wire length for a given floorplan, and may be applicable to mixed-cell placement (behaving as a post-placement step, which is left as future work).

Most floorplanning algorithms use simulated annealing to search for an optimal floorplan. The implementation of simulated annealing scheme relies on a floorplan representation where a neighbor solution is generated and examined by perturbing the representation. In the paper, we use sequence pair representation to present the approach. The reason we choose sequence pair is that it is simple and widely adopted. However, our approach is not limited to sequence pair representation. For any floorplan represented by any other presentation, we can derive a constraint graph and thus apply the approach to redistribute white space for minimizing total wire length. As we will discuss in the paper, our approach can take any input of floorplan or block placement even with a large set of additional constraints. The optimality of the approach still holds for a given floorplan topology (it does not change topology). The topology can be extracted from a floorplan/block placement, or specified by a representation such as slicing, BSG, sequence pair, TCG, O-tree, B*-tree, CBL, Q-sequence, twin binary tree and twin binary sequence.

The rest of the paper is organized as follows. Section 2 briefly reviews sequence pair and constraint graph construction to evaluate a sequence pair. Section 3 formulates the problem of minimizing total wire length, and presents a min-cost flow based approach to solve it. The capabilities of handling various constraints such as fixed-frame, IO pin, pre-placed block, boundary block, range placement, alignment and abutment, rectilinear block, soft block, cluster placement, bounded net delay, etc., are discussed in Section 4. Experimental results are reported in Section 5, followed by concluding remarks in Section 6.

2. Preliminary

A sequence pair is a pair of sequences of *n* elements representing a list of *n* blocks. The two sequences specify the geometric relations (such as left-of, right-of, below, above) between each pair of blocks as follows:

$$(...b_i...b_j...,...b_i...b_j...) \Rightarrow b_i \text{ is to the left of } b_j \quad (1)$$

$$(...b_j...b_i...,...b_i...b_j...) \Rightarrow b_i \text{ is below } b_j \quad (2)$$

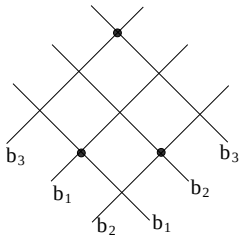


Figure 2: Sequence pair (b_3 b_1 b_2, b_1 b_2 b_3) specifies an oblique grid.

The sequence pair structure can be shown as an oblique grid (refer to Figure 2). The original paper which proposed sequence pair [14] presented an algorithm to translate a sequence pair to a placement by constructing two constraint graphs, G_h and G_v. Both G_h and G_v have $n+2$ vertices representing n blocks plus source node and sink node (representing boundaries). G_h has a directed edge (b_i, b_j) if block b_i is to the left of block b_j. Similarly, if block b_i is below block b_j, G_v has the corresponding directed edge (b_i, b_j). For any pair of blocks (e.g. b_i, b_j), there exists exactly one edge connecting the two nodes either in G_h or in G_v. Both G_h and G_v are vertex weighted, directed, acyclic graphs. The weights in G_h represent the widths of blocks, and the weights in G_v represent the heights of blocks. Given that the coordinates of a block are the coordinates of the lower-left corner of the block, a longest path algorithm can be applied to determine the coordinates of each block and the total width and height of the bounding box. As an example, the sequence pair specifying the placement in Figure 1 is (b_3 b_1 b_2, b_1 b_2 b_3), as shown in Figure 2.

3. Problem and Solution

Sequence pair specifies the topological relation between blocks. Given a sequence pair, previous algorithm compacts blocks to lower-left corner to minimize area. Even with the same minimum area, there exist different placements of blocks satisfying the topological constraint imposed by the sequence pair. It is very common that white space exists even in the floorplan packed to minimum area. The problem is to find a floorplan that fairly distributes white space and minimizes the total wire length, as defined as follows:

Problem 1. *Given a sequence pair (X,Y) with a set of m macro blocks B=$\{b_1, b_2, ..., b_m\}$ where $w_i \times h_i$ specifies the dimension of block b_i (w_i: width, h_i: height), and a set of nets $N = \{N_1, N_2, ..., N_n\}$ where $N_i, i = 1, 2, ..., n$ describes the connection between blocks, find a placement of blocks B satisfying the topological relation im-*

posed by the sequence pair, such that the total wire length

$$\sum_{i=1}^{n} \lambda_i W(N_i)$$

is minimized where $W(N_i)$ denotes the wire length of net N_i and λ_i is its weight.

Without loss of practicality, we assume that all λ_i are integers. This is true in almost all actual applications. Even in the application where some λ_i are floating numbers, we can scale them to integers. In the following we use x_i and y_i to denote the x and y coordinate of block b_i referring to the lower-left corner of the block respectively. For simple representation and easy understanding, we assume all pins are located in the center of the block. Actually as we can see later, our approach has no restriction that pins should be in the center of the block. It is common to use half perimeter of bounding box as an estimate of wire length for a net. Let us consider a net N_i connecting a set of z blocks $\{b_{i_1}, b_{i_2}, ..., b_{i_z}\}$, and use $(L_i, B_i : R_i, T_i)$ as its bounding box where (L_i, B_i) and (R_i, T_i) refer to the bottom-left and top-right corner of the bounding box respectively. Thus we have $\forall j \in \{1, 2, ..., z\}$

$$\begin{aligned} L_i &\leq x_{i_j} + w_{i_j}/2 && (3)\\ R_i &\geq x_{i_j} + w_{i_j}/2 && (4)\\ B_i &\leq y_{i_j} + h_{i_j}/2 && (5)\\ T_i &\geq y_{i_j} + h_{i_j}/2 && (6) \end{aligned}$$

Note that the coordinate $(x_{i_j} + w_{i_j}/2,\ y_{i_j} + h_{i_j}/2)$ is the center of the block b_{i_j} where pin is located. When pin is not at the center, we can use the actual pin location to substitute the coordinate. In addition, the geometric constraint imposed by sequence pair can be written as follows:

$$\begin{aligned} (...b_i...b_j..., ...b_i...b_j...) &\Rightarrow x_i + w_i \leq x_j && (7)\\ (...b_j...b_i..., ...b_i...b_j...) &\Rightarrow y_i + h_i \leq y_j && (8) \end{aligned}$$

Thus the problem can be stated as:

$$\min \sum_{i=1}^{n} \lambda_i (R_i - L_i + T_i - B_i) \quad (9)$$

subject to the set of constraints as stated in (3) (4) (5) (6) (7) (8). Since the evaluation of x and y coordinates can be done independently, the problem can be decoupled into two subproblems:

$$\min \sum_{i=1}^{n} \lambda_i (R_i - L_i) \quad (10)$$

subject to the set of constraints as stated in (3) (4) (7), and

$$\min \sum_{i=1}^{n} \lambda_i (T_i - B_i) \quad (11)$$

subject to the set of constraints as stated in (5) (6) (8). The problems (10) and (11) can be solved separately. The reason for decoupling is that it makes the algorithm faster. As we can see, all of the three problems, (9), (10) and (11), are linear programming. However, all of them have special property that all constraints are difference constraints[3]. Thus its dual problem is a min-cost flow problem, since in the constraint matrix of the dual problem, each column has exactly one "1" and "-1". As we know, linear programming is more general but much slower than min-cost flow algorithm. Let us first consider the problem (10). We can construct a network graph (called horizontal network graph) $G_H = (V_H, E_H)$ as follows.

1. $V_H = \{s, t, x_1, x_2, ..., x_m, L_1, R_1, L_2, R_2, ..., L_n, R_n\}$, where s is the source node, t is the sink node, x_i represents the x coordinate of block b_i, and L_i and R_i represent the left and right boundary of bounding box of net N_i as denoted above.

2. $E_H = \{(s, R_i) | i = 1, 2, ..., n\} \cup \{(x_i, x_j) |$ block b_i is to the right of block $b_j\} \cup \{(R_i, x_j), (x_j, L_i) |$ net N_i connects block $b_j\} \cup \{(L_i, t) | i = 1, 2, ..., n\}$, where (s, R_i) is the edge from source to right boundary of bounding box, (x_i, x_j) is the edge imposed by the sequence pair as in constraint (7), (R_i, x_j) is the edge imposed by net connection as in constraint (4), (x_j, L_i) is the edge imposed by net connection as in constraint (3), and (L_i, t) is the edge from left boundary of bounding box to sink.

3. Edge Capacity: $U_H(s, R_i) = U_H(L_i, t) = \lambda_i, \forall i \in \{1, 2, ..., n\}$; for any other edge $e \in E_H$, $U_H(e)$ is unlimited.

4. Cost Function: $C_H(s, R_i) = 0$, $C_H(L_i, t) = 0$, $C_H(x_i, x_j) = -w_j$, $C_H(R_i, x_j) = -w_j/2$, and $C_H(x_j, L_i) = w_j/2$.

It should be noted that the subgraph, which contains only the vertices $x_i, i = 1, 2, ...m$ and the edges (x_i, x_j) imposed by sequence pair, is similar to the horizontal constraint graph mentioned in [14]. The difference is that the direction of edges is inverted and the edge cost is negative. Thereafter, in [14] a longest path algorithm is applied to compute the positions of blocks, while in the paper we shall use min-cost flow algorithm in the sense of shortest path. It should also be noted that the transitive edges on the subgraph can be safely omitted, which will speed up the computation considerably.

Thus we compute the min-cost flow of amount $\sum_{i=1}^{n} \lambda_i$ on the graph G_H, which solves the dual problem. Our goal is to compute the positions of blocks subject to the constraints and minimize the total wire length (the primal problem), which can be done as follows. We first compute the residual graph derived from the min-cost flow. Then a shortest path algorithm applied on the residual graph would give the positions for all blocks. If necessary, a common source node connecting to all other nodes can be added to the residual graph for shortest path computation. [1]

Analogously, we can construct another network graph and solve the problem (11) by min-cost flow approach. The graph (called vertical network graph) is denoted as $G_V = (V_V, E_V)$.

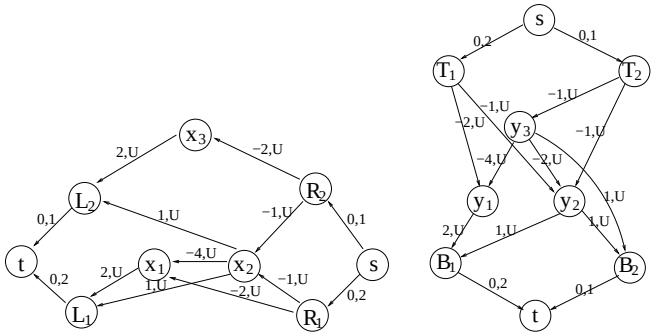
[1] Actually, the results of positions are the node potentials (or called "price"). Many of min-cost flow algorithms compute both edge flows and node potentials at the same time. Thus in this case, the steps of deriving residual graphs and applying shortest path algorithm to compute positions can be skipped.

We use the example as shown in Figure 1 to illustrate the approach. The input of the problem is: sequence pair (b_3 b_1 b_2, b_1 b_2 b_3) with 3 blocks, and nets $N_1 = \{b_1, b_2\}$ with weight $\lambda_1 = 2$, $N_2 = \{b_2, b_3\}$ with weight $\lambda_2 = 1$. Then the problem (10) to minimize wire length in x dimension can be stated as follows:

$$\min\{2(R_1 - L_1) + (R_2 - L_2)\}$$

subject to

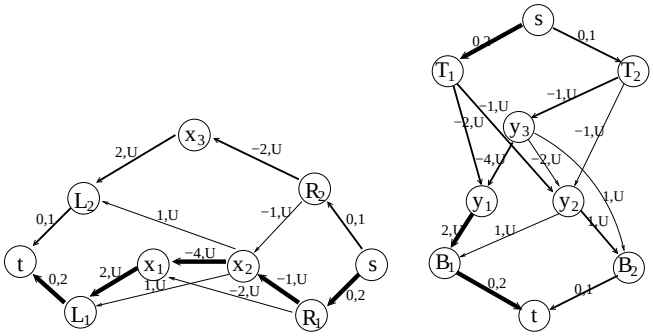
$$\begin{aligned}
x_1 + 4 &\leq x_2 \\
x_1 + 2 &\geq L_1 \\
x_1 + 2 &\leq R_1 \\
x_2 + 1 &\geq L_1 \\
x_2 + 1 &\leq R_1 \\
x_2 + 1 &\geq L_2 \\
x_2 + 1 &\leq R_2 \\
x_3 + 2 &\geq L_2 \\
x_3 + 2 &\leq R_2
\end{aligned}$$



(a) (b)

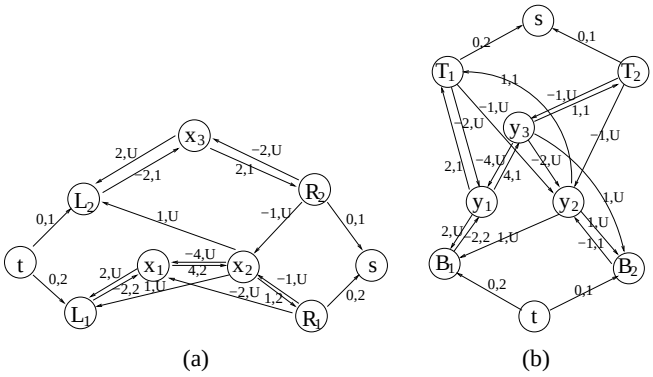
Figure 3: (a) The horizontal network graph. (b) The vertical network graph. The pair of numbers, "c,u", on the edge represent cost and capacity respectively, and "U" means unlimited capacity (the same meaning on the graphs that follow).

Then this can be transformed to min-cost flow problem in the network graph G_H as shown in Figure 3(a). Similarly, the problem (11) to minimize wire length in y dimension is transformed to min-cost flow problem in the network graph G_V as shown in Figure 3(b). Then we compute the min-cost flow of amount: 3 (because $\lambda_1 + \lambda_2 = 3$) on the two graphs, G_H and G_V. The results are illustrated in Figure 4(a) and Figure 4(b) respectively. Based on the flow results, we derive the residual graphs of G_H and G_V, as shown in Figure 5(a) and Figure 5(b) respectively. Then we apply shortest path algorithm on the residual graphs to compute the positions of blocks by adding a common source node connecting all other nodes. Thus the results are: $x_1 = -5$, $x_2 = -1$, $x_3 = -2$, $y_1 = -5$, $y_2 = -4$, and $y_3 = -1$. The placement with minimum wire length is shown in Figure 6. The overall approach is summarized as follows.



(a) (b)

Figure 4: (a) The min-cost flow on the horizontal network graph. (b) The min-cost flow on the vertical network graph. The highlighted lines represent flows. Their widths are proportional to the amount of flow.



(a) (b)

Figure 5: (a) The horizontal residual graph. (b) The vertical residual graph.

Algorithm Min-wire

1. Construct the network graphs G_H and G_V
2. Apply min-cost flow algorithm on G_H and G_V
3. Derive the residual graphs of G_H and G_V
4. Apply shortest path algorithm on residual graphs to compute positions of blocks

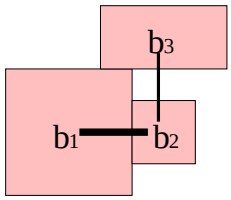


Figure 6: The placement with minimum wire length.

The min-cost flow based algorithm, Min-wire, optimally solves the problem of minimizing total wire length, as stated in the following theorem.

Theorem 1. *The algorithm, Min-wire, generates a placement of*

all blocks such that the total wire length is minimized optimally for the given sequence pair.

Note that we show here how to obtain minimum wire length. The capability of minimizing wire length and meanwhile keeping minimum area (with a fixed-frame constraint) will be discussed in the next section.

The complexity of the algorithm, Min-wire, is determined by min-cost flow, since other steps are smaller portions compared to min-cost flow algorithm. Finding a min-cost flow in a network is a classical problem for which several polynomial-time optimal algorithms are available [3]. The number of vertices in either G_H or G_V is $O(m+n)$ where m is the number of blocks and n is the number of nets. The number of edges on the subgraph, which contains only the vertices representing blocks and the edges between, is $O(m \log m)$ on average[11]. The rest of edges includes the edges introduced by net connections, and the edges incident from/to source/sink. The number of edges incident from source and to sink is $O(n)$. The edges introduced by net connections in the graph is proportional to the number of pins in all nets. Typically in practice, we can assume that the number of pins is a constant on average in a net. Thus the number of edges introduced by net connections is typically $O(n)$. In total, the number of edges is $O(m \log m + n)$. Therefore, if we adopt Orlin's algorithm in [3] to compute the min-cost flow, the time complexity of the algorithm Min-wire is typically $O(|E|\log|V|(|E|+|V|\log|V|)) = O((m\log m+n)\log(m+n)(m\log m+n+(m+n)\log(m+n))) = O((m\log m+n)(m+n)\log^2(m+n))$. Practically, we can assume that net weight λ_i is $O(1)$ (for example, 1-10), which is true in most applications. We observe that too large weight is unnecessary in actual applications. When net weight is beyond some threshold, it behaves the same in minimizing wire length. Then we can apply successive shortest path augmenting algorithm in computing min-cost flow[3] which is faster in the case. Thus, the complexity is $O(nS(m,n))$ where $S(m,n)$ denotes the time taken to solve a shortest path problem. We can associate each node i with a distance label $d(i)$. Then the "reduced" cost for edge (i,j) is defined as $C'(i,j) = C(i,j) + d(i) - d(j)$. Since in the shortest path computation, it is always true that $d(j) \le d(i) + C(i,j)$, thus $C'(i,j) \ge 0$. Therefore the graph with "reduced" costs has no negative cost edge. We can get initial $d(i)$ using Bellman-Ford algorithm. After that, we can apply Dijkstra algorithm to the graph with "reduced" costs, since the "reduced" costs $C'(i,j) \ge 0$. Note that Dijkstra algorithm is faster but can not handle negative cost. Thus $S(m,n)$ is the complexity of Dijkstra algorithm. Finally the complexity is $O(n(|E|+|V|\log|V|)) = O(n(m\log m+n+(m+n)\log(m+n))) = O(n(m+n)\log(m+n))$.

4. Discussion of Capabilities

It is useful and important in applications that floorplanning handles constraints[25]. As we can see, the approach is capable of handling various constraints without loss of optimality.

4.1 Fixed-frame

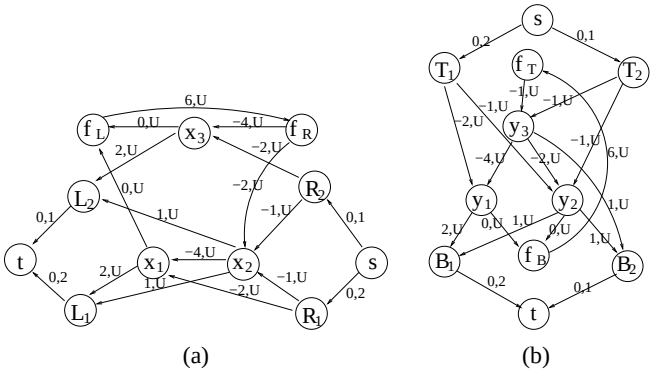


Figure 7: The modification of graphs to handle fixed-frame where the frame is 6×6. (a) The horizontal network graph. (b) The vertical network graph.

In some applications, floorplanning is confined in a given frame, $W \times H$, where W and H represent width and height respectively. In addition, if we still want to keep the minimum area in minimizing wire length, we can solve the problem with a frame of minimum area. When a frame is taken into account, we modify the graphs as follows. To horizontal network graph G_H, two nodes, f_L and f_R, are added where f_L and f_R represent the left and right boundary of the frame respectively. We have $x_i + w_i \le f_R$, $x_i \ge f_L$ and $f_R - f_L \le W$. Accordingly, a set of edges are added, (f_R, x_i) with cost $-w_i$ and unlimited capacity, (x_i, f_L) with cost 0 and unlimited capacity, $i = 1,2,...,m$, and (f_L, f_R) with cost W and unlimited capacity. Again, the transitive edges in (f_R, x_i) and (x_i, f_L) can be omitted. Similarly, two nodes, f_B and f_T representing the bottom and top boundary of the frame respectively, and the corresponding edges are added to vertical network graph G_V. Figure 7 illustrates the two modified graphs for the example above. The frame is 6×6 (minimum area). Thus the algorithm Min-wire can still be applied to minimize the total wire length and place blocks in the given frame. Note that the node f_R (f_T) will act as the source node in the step of shortest path computation in G_H (G_V) to obtain the positions of blocks. In the computations, we assign $f_R = W$ and $f_T = H$. Figure 1(b) actually gives the optimal placement within the frame.

4.2 Composite Cost Function

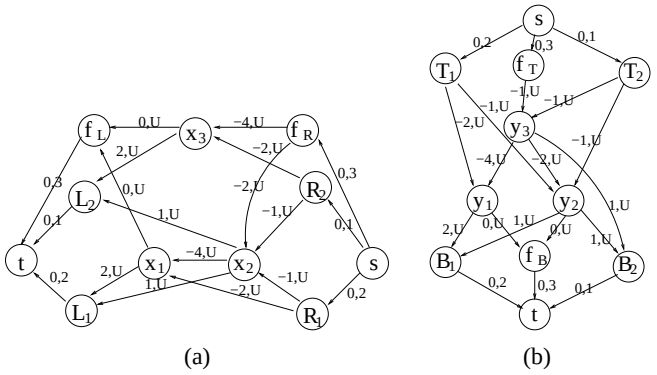
We have talked about fixed-frame constraint that blocks are confined within a given frame. Actually, the method is an exact algorithm to optimize the composite cost function of area and wire length:

$$\min\{\alpha(W+H) + \sum_{i=1}^{n} \lambda_i W_i\}$$

Note that existing methods can only minimize area, and use compacted blocks' locations to compute the cost of wire length. By introducing f_L, f_R, f_B, f_T as in handling fixed-frame constraint, we can convert the objective function to:

$$\min\{\alpha(f_R - f_L + f_T - f_B) + \sum_{i=1}^{n} \lambda_i (R_i - L_i + T_i - B_i)\}$$

Note that W and H become variables and $W = f_R - f_L, H = f_T - f_B$. The network-flow-based approach can still be used to optimize the composite cost, by modifying the graphs, G_H and G_V.



(a) (b)

Figure 8: The modification of graphs to minimize composite cost where $\alpha = 3$. (a) The horizontal network graph. (b) The vertical network graph.

It is similar to the modification that handles fixed-frame constraint, except that there is no edge (f_L, f_R) on graph G_H and no edge (f_B, f_T) on graph G_V. Instead, we add edges, (s, f_R) with cost 0 and capacity α and (f_L, t) with cost 0 and capacity α, to graph G_H; and we add edges, (s, f_T) with cost 0 and capacity α and (f_B, t) with cost 0 and capacity α, to graph G_V. Figure 8 illustrates the two modified graphs for the example above to optimize the composite cost function, where $\alpha = 3$. Thus the min-cost flow based algorithm can be used to optimally minimize the composite cost. It should be noted that the amount of flow is $\alpha + \sum_{i=1}^{n} \lambda_i$ in the case.

4.3 Handling IO Pins

Usually there exist IO nets which connect to pins on the boundary of the frame (IO pins or IO pads). Let us consider a net N_i connects an IO pin at location (p_x, p_y). Thus $L_i \le p_x \le R_i$ and $B_i \le p_y \le T_i$. Assume the frame is $W \times H$. We let $f_R = W$ and $f_T = H$. Then equivalently, $L_i - f_R \le p_x - W \le R_i - f_R$ and $B_i - f_T \le p_y - H \le T_i - f_T$. As a result, we add two edges, (f_R, L_i) (with cost $p_x - W$ and unlimited capacity) and (R_i, f_R) (with cost $W - p_x$ and unlimited capacity), to graph G_H, and add two edges, (f_T, B_i) (with cost $p_y - H$ and unlimited capacity) and (T_i, f_T) (with cost $H - p_y$ and unlimited capacity), to graph G_V. Thus the algorithm Min-wire can be applied. In this way, fixed pin can be handled where the pin location may not be on the boundary of the frame.

4.4 Pre-placed and Boundary Blocks

In the case when some blocks are to be placed at fixed location or on the boundary of the frame, our algorithm should be applied to a graph with additional edges. For example, a block b_i is placed at a location (l_x, l_y), i.e. $x_i = l_x$ and $y_i = l_y$. Thus $x_i - f_R = l_x - W$ and $y_i - f_T = l_y - H$. Equivalently, $x_i - f_R \le l_x - W$, $x_i - f_R \ge l_x - W$, $y_i - f_T \le l_y - H$, and $y_i - f_T \ge l_y - H$. These are transformed to edges (f_R, x_i) and (x_i, f_R) on graph G_H, and edges (f_T, y_i) and (y_i, f_T) on graph G_V. Boundary blocks can be handled similarly in the sense that boundary blocks fix locations in x or y coordinate.

4.5 Range Placement

Range constraint specifies that a block is to be placed within a given range. Pre-placed constraint is a special case of range constraint. Similarly, we can add additional edges to graph G_H and G_V to enforce the computation of position in algorithm Min-wire such that the block is placed within the range.

4.6 Alignment and Abutment

Alignment constraint specifies several blocks to be aligned in a row within a range[19]. It can be transformed to a set of difference constraints that keep the relative positions between them. Thus we can add additional edges to the graphs accordingly. Abutment is a special case of alignment.

4.7 Rectilinear Blocks

A rectilinear block is partitioned into a set of rectangular subblocks. Then a set of constraints are used to keep the relative positions, which can be transformed to the additional edges in the graphs accordingly[7].

4.8 Soft Blocks

In floorplanning, the shapes of some blocks may not be fixed. For example, the areas of some blocks are fixed, but their width/height ratio can be changed in some range. These kinds of blocks are called soft blocks. To handle soft blocks, a move (perturbation) can be introduced in simulated annealing to change the shape of soft block, as in [9]. In this way, the move is random. In another way (more intelligent) as in [1, 21], each time a soft block on a critical path ("critical" means that the path determines the size of the chip) is selected and its shape is optimized. Our approach can be applied to minimize wire length in both ways.

4.9 Cluster Placement

It is useful in applications that several blocks are placed close to each other (cluster placement). In other words, the distance between any two of the blocks should not be too far away. This can be written as a set of constraints that specify the distance bound between any two of the blocks. One-dimensional cluster placement specifies the distance bound either in x or in y dimension, which can be written as a set of difference constraints. Thus we can solve the problem by adding the corresponding edges to the graphs.

In general case of two-dimensional cluster placement, the distance bound is specified on the summation in x and y dimensions. We can use a heuristics to break two-dimensional cluster placement proportionally into two one-dimensional cluster placement problems in x and y dimensions, based on compaction result or original block placement. In other way, we can also solve the problem using Lagrangian relaxation. We treat the cluster placement as a "virtual" net, and adjust the weight of the net (Lagrangian multiplier) to make it satisfy the distance bound. Thus each iteration of Lagrangian relaxation is still to solve a min-cost flow problem.

4.10 Bounded Net Delay

The approach is to minimize the total wire length, which can not guarantee bounded delay for critical nets. To address bounded net delay, we use a linear function in terms of distance to estimate delay. Although interconnect delay is quadratic in terms of wire length, with appropriate buffer insertions the actual delay is close to linear in terms of source-sink distance. In this way we convert bounded net delay into bounded net wire length. Thus as in [19], we impose constraints on the bounding box of the net, which results in the additional edges in the graphs accordingly.

We have the following necessary and sufficient condition with respect to all these constraints.

Theorem 2. *There exists a feasible placement that satisfies all these constraints if and only if there is no negative cycle in graphs G_H and G_V.*

When a graph has a negative cycle with unlimited capacity, there does not exist min-cost flow. As we can see, in the graph G_H and G_V, the edges except the edges incident from source node or to sink node have unlimited capacity, and the edges incident from source node or to sink node can not be part of any cycle. Thus any negative cycle will have unlimited capacity. If there is no negative cycle, then the algorithm Min-wire can be used to compute a placement that satisfies all constraints and has the minimum wire length.

Although the condition is similar to that in [7], there exist important differences. (i) The graphs are different. The graph in [7] contains only nodes representing blocks/subblocks. (ii) The approach in [7] operating on its graph thus does area packing only, while our approach can minimize both area and wire length. (iii) Longest path algorithm is used in [7], while longest/shortest path can not solve our problem and instead main part of our algorithm is min-cost flow.

4.11 Applications to Block Placement

Although we take input of sequence pair in the problem definition, the approach can be applied to any floorplan/block placement. Given a floorplan/block placement, we can first extract the topological relation for any pair of blocks and describe as "left of"/"below". For the pair with diagonal relation, we can choose one of "left of"/"below" based on which of the two distances in x and y dimension is longer. This is because choosing the longer one gives us more freedom in moving blocks around. For example, in Figure 1(a), we specify that block b_2 is below block b_3, rather than block b_3 is to the left of block b_2. Then we can construct the constraint graphs and network graphs. Thereafter, we can apply the approach to minimize wire length.

For the floorplan specified by representation other than sequence pair (such as slicing, BSG, TCG, CBL, Q-sequence, twin binary tree and twin binary sequence), we can also construct constraint graphs which are equivalent to the topology specified by the representation. Thus the approach can still be applied. Note that the approach does not change the topology. The topology information in O-tree and B*-tree is incomplete (only x-dimension relation is specified and floorplan is obtained by packing). However, we can derive block placement from O-tree and B*-tree and then build constraint graphs from the placement. We summarize the result as follows.

Theorem 3. *The algorithm, Min-wire, is optimal in minimizing total wire length for a given floorplan topology. The topology can be extracted from a floorplan/block placement, or specified by any floorplan representation.*

5. Experimental Results

We have implemented the algorithm and integrated with the floorplanner, FAST-SP[18]. Our program can also read an existing floorplan and redistribute white space to optimize wire length. Assuming that $\lambda_i = O(1)$, we use successive shortest path augmenting algorithm in min-cost flow computation.

We have tested the program as a post-floorplanning step with two floorplanner, FAST-SP[18] and Parquet 3[1]. The test problems are derived from MCNC benchmarks for block placement. We first run FAST-SP or Parquet 3 to obtain a floorplan with option of "minimize wire length". Note that both FAST-SP and Parquet 3 compact blocks to the left and bottom. Then the algorithm Min-wire is applied to further optimize wire length. For all tests, we use the center of block as pin's location, and impose a fixed frame constraint. The locations of IO pins (IO pads) are resized proportionally to the frame boundaries. Table 1 lists the experimental results for minimizing wire length, where all blocks are hard blocks. It should be noted that our program does not change floorplan topology and area. Thus area is omitted from the table. The experiments were carried out on a laptop of Pentium 4 Mobile(2.4Ghz). As we can see, the algorithm is very efficient in that it takes less than 0.4 seconds for all of the benchmarks. It is also very effective in that it can further improve the wire length of even very compact floorplans by 4.2% on average. As illustrations, Figure 9 and 10 display the placement results of original and after optimization for ami33 and ami49 in FAST-SP respectively.

6. Concluding Remarks

In the paper, we have presented a novel method to minimize wire length in floorplanning. The method optimally distributes white space among blocks and guarantees to obtain the minimum total wire length for a given floorplan topology. It is also an exact algorithm to optimize the composite cost function of area and wire length: $\min\{\alpha(W+H)+\sum_{i=1}^{n}\lambda_i W_i\}$. We have also shown that the method can handle various constraints such as fixed-frame, IO pins, pre-placed blocks, boundary blocks, range placement, alignment and abutment, rectilinear blocks, soft blocks, one-dimensional cluster placement, and bounded net delay, without loss of optimality. Experimental results show it is very efficient and effective. Thus it provides an ideal way to refine floorplanning (post-floorplanning). The future work is to extend the method to consider routing congestion and buffer insertion in floorplanning and to apply it in mixed-cell placement.

Table 1: Results of improving wire length in post-floorplanning of FAST-SP and Parquet 3.

circuit	block	net	FAST-SP				Parquet 3			
			wire(*mm*)		improve	time(s)	wire(*mm*)		improve	time(s)
			original	after			original	after		
apte	9	97	426.7	418.9	1.8%	0.02	476.3	458.0	3.8%	0.03
xerox	10	203	486.1	462.8	4.8%	0.07	581.6	550.6	5.3%	0.06
hp	11	83	170.0	161.5	5.0%	0.02	161.1	152.4	5.4%	0.02
ami33	33	123	60.0	58.0	3.3%	0.03	77.2	74.1	4.0%	0.03
ami49	49	408	790.1	760.2	3.8%	0.38	857.8	818.9	4.5%	0.36

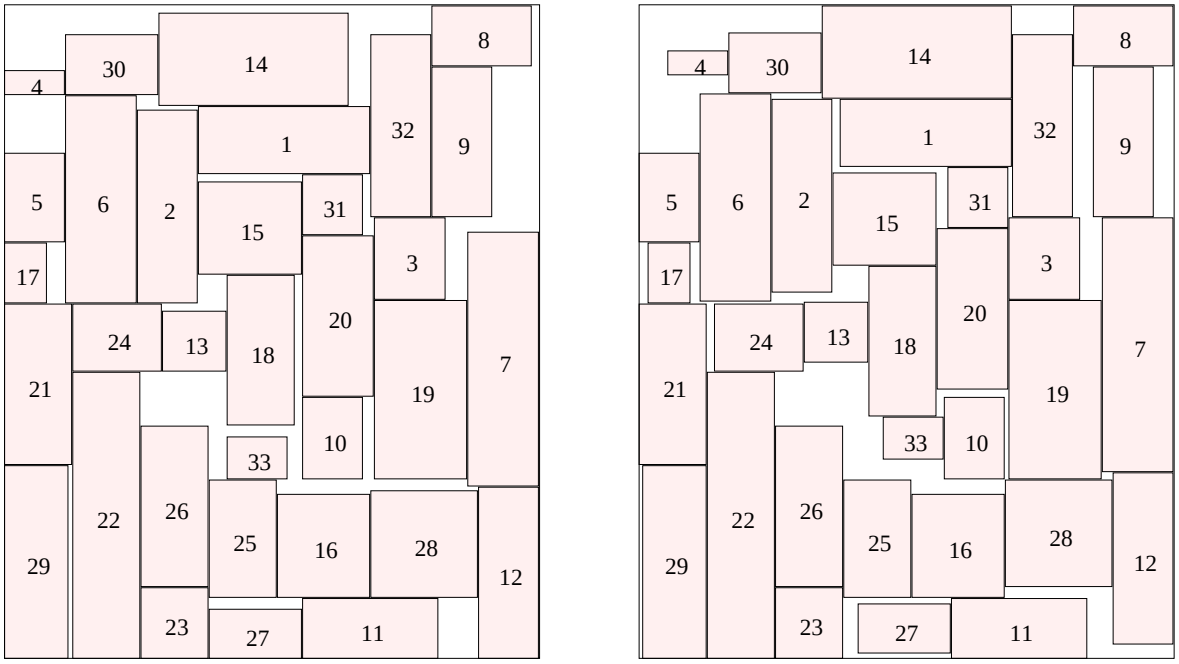


Figure 9: (a) The original ami33 placement in FAST-SP. (b) The new placement result when minimizing wire length in the same frame.

7. References

[1] S.N. Adya and I.L. Markov. "Fixed-outline floorplanning through better local search", ICCD-01, pp. 328-334, 2001.

[2] S.N. Adya, I.L. Markov, and P.G. Villarrubia. "On whitespace and stability in mixed-size placement and physical synthesis", ICCAD'03, pp. 311-317, 2003.

[3] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. Network Flows theory, algorithms, and applications, Prentice Hall Inc., New Jersey, 1993.

[4] C.J. Alpert, G.J. Nam, and P.G. Villarrubia. "Free space management for cut-based placement", ICCAD'02, pp. 746-751, 2002.

[5] Y.C. Chang, Y.W. Chang, G.M. Wu, and S.W. Wu. "B*-trees: a new representation for non-slicing floorplans", DAC-2000, pp. 458-463, 2000.

[6] K. Doll, F.M. Johannes, K.J. Antreich. "Iterative placement improvement by network flow methods", IEEE Trans. on CAD, vol 13:10, pp. 1189-1199, 1994.

[7] K. Fujiyoshi, and H. Murata. "Arbitrary convex and concave rectilinear block packing using sequence pair", ISPD-99, pp. 103-110, 1999.

[8] P.N. Guo, C.K. Cheng, and T. Yoshimura, "An O-tree representation of non-slicing floorplans and its applications", DAC-99, pp. 268-273, 1999.

[9] X. Hong, G. Huang, Y. Cai, J. Gu, S. Dong, C.K. Cheng, and J. Gu. "Corner block list: an effective and efficient topological representation of non-slicing floorplan", ICCAD-00, pp. 8-12, 2000.

[10] A.B. Kahng, P. Tuckler, and A. Zelikovsky. "Optimization of linear placements for wirelength minimization with free sites", ASP-DAC'99, pp. 241-244, 1999.

[11] C. Lin. "Incremental mixed-signal layout generation concepts", phd thesis, 2002.

[12] J.M. Lin and Y.W. Chang. "TCG: a transitive closure graph-based representation for non-slicing floorplans", DAC-01, pp. 764-769, 2001.

[13] Y. Ma, X. Hong, S. Dong,S. Chen, Y. Cai, C.K. Cheng, and J. Gu, "An integrated floorplanning with an efficient buffer planning algorithm", ISPD'03, pp. 136-142, 2003.

[14] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. "VLSI module placement based on rectangle-packing by the sequence pair", *IEEE Transaction on Computer Aided Design of Integrated Circuits and*

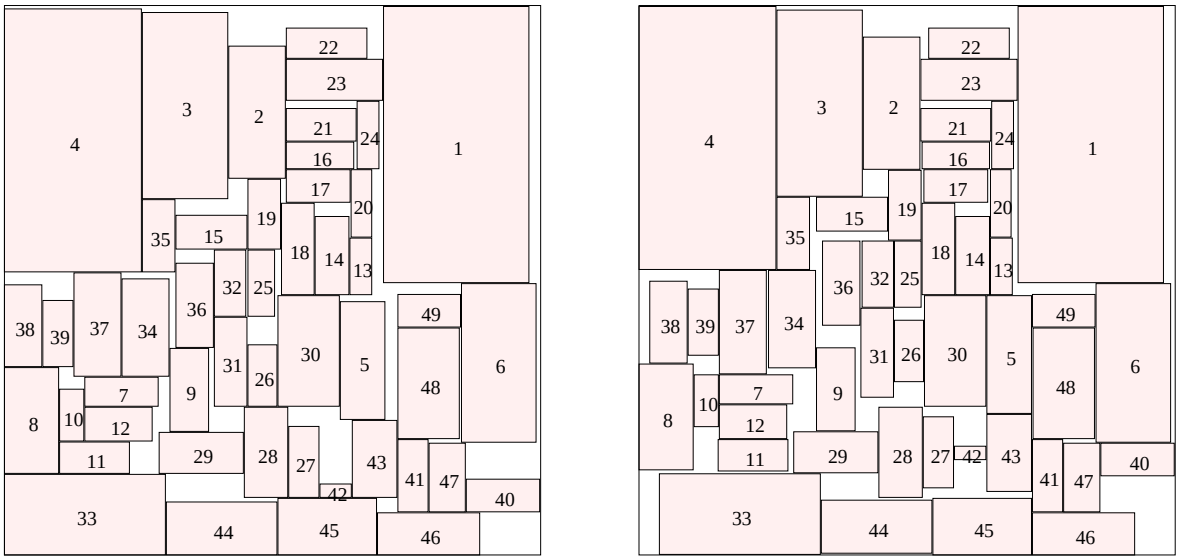


Figure 10: (a) The original ami49 placement in FAST-SP. (b) The new placement result when minimizing wire length in the same frame.

Systems, vol. 15:12, pp. 1518-1524, 1996.

[15] S. Nakatake, H. Murata, K. Fujiyoshi, and Y. Kajitani. "Module placement on BSG-structure and IC layout applications", ICCAD-96, pp. 484-491, 1996.

[16] R.H.J.M. Otten. "Automatic floorplan design", DAC-82, pp. 261-267, 1982.

[17] K. Sakanushi and Y. Kajitani. "The quarter-state sequence (Q-sequence) to represent the floorplan and applications to layout optimization", IEEE APCCAS, pp. 829-832, 2000.

[18] X. Tang and D.F. Wong. "FAST-SP: A fast algorithm for block placement based sequence pair", ASPDAC-01, pp. 521-526, 2001.

[19] X. Tang and D.F. Wong. "Floorplanning with alignment and performance constraints", DAC-02, pp. 848-853, 2002.

[20] D.F. Wong and C.L. Liu. "A new algorithm for floorplan design", DAC-86, pp. 101-107, 1986.

[21] H. Xiang, X. Tang and D.F. Wong. "Bus-driven floorplanning", ICCAD-03, pp. 66-73, 2003.

[22] X. Yang, B.K. Choi and M. Sarrafzadeh. "Routability driven white space allocation for fixed-die standard cell placement", ISPD'02, pp. 42-50, 2002.

[23] B. Yao, H. Chen, C.K. Cheng, and R. Graham. "Revisiting floorplan representation", ISPD-01, pp. 138-143, 2001.

[24] F.Y. Young, C.N. Chu, and Z.C. Shen. "Twin binary sequences: a non-redundant representation for general non-slicing floorplan", ISPD-02, pp. 196-201, 2002.

[25] F.Y. Young, C.N. Chu, and M.L. Ho. "Placement constraints in floorplan design", IEEE Trans. on VLSI, vol. 12:7, pp. 735-745, 2004.