Unified Arbitrary Rectilinear Block Packing and Soft Block Packing Based on Sequence Pair

Huaizhi Wu
Wayne Dai

UCSC-CRL-00-07
May 4, 2000

Jack Baskin School of Engineering
University of California, Santa Cruz
Santa Cruz, CA 95064 USA

ABSTRACT

To the best of our knowledge, this is the first algorithm unifying arbitrary rectilinear block packing and soft block packing. Furthermore, this algorithm handles arbitrary convex or concave rectilinear block packing in the most efficient way compared to other sequence pair-based approaches. At the same time, the algorithm can handle rectangle soft block effectively. The concept of non-redundant constraint graph together with its algorithms play critical role in unifying the arbitrary rectilinear block packing and soft block packing. This general block packing tool builds the foundation for floorplanning with IP reuse. The experimental results demonstrate the efficiency and effectiveness of this general block packing.

Keywords: floorplan, block packing, soft block, rectilinear block, sequence pair, non-redundant constraint graph, related-vertices-grouped constraint graph, feasible slack, bottleneck path, simulated annealing.
I. Introduction

Due to the rapid scaling of circuit size and increasing complexity of IC design, design reuse has become of great interest of the design community. As the IP blocks to be reused are not often of rectangle shapes, good packing algorithm for floorplans with arbitrary rectilinear (hard) blocks is desired. On the other hand, as many blocks have not been detailedly designed in the early floorplanning stage, shape flexibility is to be exploited to improve the floorplanning quality, and good algorithm for floorplans with soft (rectangle) block is demanded. Several works have been done in each of the two areas ([1], [2], [3] for arbitrary rectilinear block packing; [4], [5] for soft block packing with general non-slicing structure), but no method accommodating both of the two aspects have been proposed so far. By addressing this problem, this paper makes the following major contributions: (1) It proposes a new technique for evaluating sequence pair; (2) It re-designs the rectilinear block packing algorithm and soft block packing algorithm and unifies them together, based on some existing techniques and more importantly, several new observations of the rectilinear/soft block packing properties; (3) It extends the existing stochastic moves to be more globally.

The following of this paper is organized as follows: Section II briefly introduces the sequence pair structure. Section III describes in details the new method of evaluating SP. Section IV and section V describe the rectilinear block packing and soft block packing algorithms, respectively. Section VI describes the extended stochastic moves. Section VII presents the experimental results and section IX gives some concluding remarks.

II. Sequence Pair (SP) Structure

A sequence pair for a set of \( n \) rectangle blocks is a pair of permutations of the \( n \) block names [6]. For example, \((d b a e f c, a b c d e f)\) is a sequence pair of the block set \(\{a, b, c, d, e, f\}\).

The topological constraint between every pair of blocks \( x, y \) is defined as follows:

**H-constraint**: \((.. x .. y ..,.. x .. y ..) \Rightarrow x \text{ is left of } y\)

**V-constraint**: \((.. y .. x ..,.. x .. y ..) \Rightarrow x \text{ is below } y\)

Given a sequence pair, a compacted packing of the blocks can be obtained by using the directed acyclic H/V-constraint graphs \( G_h / G_v \) which are constructed faithfully to the H/V-constraint described above. That is to say, for every pair of blocks \( x, y \), if \( x \) is left of \( y \), we add an edge \((x, y)\) to \( G_h \); if \( x \) is below \( y \), we add an edge \((x, y)\) to \( G_v \). Moreover, there is a source \( s_h \) / \( s_v \) connected to each leftmost / lowest block and a sink \( t_h \) / \( t_v \) connected to each rightmost / upmost block in \( G_h / G_v \). Fig 1 shows an example. The weight of the each vertex in \( G_h / G_v \) is the width / height of the corresponding block. The X/Y-coordinate of each block can be determined by the length of the longest path from the source \( s_h / s_v \) to the corresponding vertex in \( G_h / G_v \).

![Fig 1 Constraint Graphs of Sequence Pair](image-url)
III. Non-redundant Constraint Graph

Sequence Pair defines a “left/right” or “below/above” relation between every pair of blocks. So the union of the complete horizontal constraint graph $G_h$ and the complete vertical constraint graph $G_v$ has $C(n, 2) = n(n-1)/2$ edges. A quick glance at the graphs in Fig 1 shows that some edges (e.g. $(a, f)$ in $G_h$, $(a, d)$ in $G_v$) are redundant. In general, a horizontal edge between two vertices $v_i$ and $v_j$ does not need to be present if there is another path from $v_i$ and $v_j$ passing one or more other vertices. And we define such edge as redundant edge. Specifically, if there is an edge from $v_i$ to $v_j$ and from $v_j$ to $v_i$, then the left/right constraint between $v_i$ and $v_j$ is implicitly endured by the transition. So the edge $(v_i, v_j)$ can be removed without changing any constraint defined by the SP. A constraint graph with all redundant edges removed is called a non-redundant constraint graph.

As the time of calculating the longest path for each block is linear to the number of edges in the graph, removing those redundant edges will result in a significant reduction of time in evaluating a constraint graph.

The constraint graph $G_h$ and $G_v$ are subsets of $G_h^*$ and $G_v^*$ simultaneously by finding all the immediate left-predecessors and below-predecessors for each block. As we know, if block $a$ is left of or below block $b$, $a$ precedes $b$ in $\Gamma_+$. Therefore, for each block $b$, we only need to consider the blocks preceding $b$ in $\Gamma_+$. And naturally we should process all the blocks in the order given by $\Gamma_+$. Without loss of generality, we assume that $\Gamma_+ = (a_1, a_2, \ldots, a_k)$. Then the algorithm goes as follows:

First we see how to get a pair of redundancy-reduced constraint graph $G_h^*$ and $G_v^*$ from a given SP in the time linear to the total number of edges in $G_h^*$ and $G_v^*$.

**Lemma 1** The constraint graph $G_h^*$ and $G_v^*$ output by algorithm RRCG is equivalent to the SP which it is built from in terms of topological constraints.

**Proof** As $G_h^*$ and $G_v^*$ are subsets of $G_h$ and $G_v$, respectively, $G_h^*$ and $G_v^*$ do not alter any topological constraint defined by the corresponding SP. If we can prove that $G_h^*$ and $G_v^*$ do not reduce any topological constraint defined by the corresponding SP either, the proof is obviously done. For the latter, we only need to show that for any block $a_i$, it’s reachable from any of the blocks preceding it in $\Gamma_+$ in either $G_h^*$ or $G_v^*$, which means the topological relation between any pair of blocks is given by $G_h^*$ and $G_v^*$.

Without loss of generality, we assume that $a_{i-1}$ is left of $a_i$, as shown in Fig 3(a). All blocks preceding $a_i$ in $\Gamma_+$ are either in region I or in region II of the constraint graph. As all blocks in region I can reach $a_{i-1}$ in $G_h^*$, they can also reach $a_i$ in $G_h^*$ through the edge $(a_{i-1}, a_i)$. Let us consider blocks in region II. If $a_{i-1}$ has no predecessor in $G_v^*$, then region II contains no block, we are done. Otherwise, let $a_k$ be the predecessor of $a_{i-1}$ in $G_v^*$ with largest index in $\Gamma_+$. $a_k$ is either left of or below $a_i$.

![Fig 2 Algorithm RRCG](#)

**Lemma 1** The constraint graph $G_h^*$ and $G_v^*$ output by algorithm RRCG is equivalent to the SP which it is built from in terms of topological constraints.
(i) \( a_k \) is left of \( a_i \) : As shown in Fig 3(b), there is an edge \((a_k, a_i)\) in \(G_h \)' according to the algorithm RRCG. All blocks in region II(a) can reach \( a_k \) in \(G_h \)', so they can also reach \( a_i \) in \(G_h \)' through the edge \((a_k, a_i)\). Region II(b) must contain no block, otherwise \( a_k \) can not be the predecessor of \( a_{i-1} \) in \(G_v \) with largest index in \(\Gamma_\cdot\). We are left with Region II(c). The problem is reduced to a smaller size than that in Fig 3(a).

(ii) \( a_k \) is below \( a_i \) : As shown in Fig 3(c), there is an edge \((a_k, a_i)\) in \(G_v \)' according to the algorithm RRCG. Again, region II(b) contains no block. All blocks in region II(c) can reach \( a_k \) in \(G_v \)' so they can also reach \( a_i \) through the edge \((a_k, a_i)\). Now for region II(a). If \( a_k \) has no predecessor in \(G_h \)', then region II(a) contains no block, we are done. Otherwise, let \( a_j \) be the predecessor of \( a_k \) in \(G_h \)' with largest index in \(\Gamma_\cdot\), as shown in Fig 3(d). Again, \( a_j \) is either left of or below \( a_i \). For the former case, we are left with region II(a)-1; for the latter case case, we are left with region II(a)-2. In both case the problem is reduced to a smaller size than that in Fig 3(c).

Therefore by deduction we can prove that any block preceding block \( a_i \) in \(\Gamma_\cdot\) can reach it either by a horizontal path in \(G_h \)' or by a vertical path in \(G_v \).

By the algorithm RRCG, we can obtain a pair of graphs \(G_h \)' and \(G_v \)' equivalent to but smaller than \(G_h \) and \(G_v \). But it may still contain some redundant edges in the case shown in Fig 4. Block \( f \) is left of block \( h \); block \( c \) is \( f \)'s rightmost below-predecessor in \(G_v \)' and \( c \) is below \( h \); \( b \) is \( c \)'s upmost left-predecessor in \(G_h \). As \( b \) may be left of \( f \), in which case \( b \) can reach \( h \) through the path from \( b \) to \( f \) and the the edge \((f, h)\), so edge \((b, h)\) may be a redundant edge.
To avoid adding an edge \((b, h)\) in \(G_h^*\) in such case, we modify the algorithm RRCG and get the non-redundant constraint graph \(G_h^*\) and \(G_v^*\) as follows:

1. \(a_l\) has no predecessor in \(G_h^*\) or \(G_v^*\);
2. Suppose we are done for all \(a_k\) \((k < i)\). For \(a_i\):
   i. Let \(j = i - 1\), last_dir = no_def, last_blk = no_def;
   ii. Compare \(\Gamma_+ (a_i)\) and \(\Gamma_+ (a_j)\) to decide whether \(a_i\) is left or below \(a_j\), add an edge \((a_j, a_i)\) to \(G_h^*\) or \(G_v^*\), and record the direction of this iteration as ‘h’ or ‘v’ accordingly. If the direction of last iteration is not ‘no_def’ and is different from the direction of this iteration, then modify the reference-block to be the \(a_i\) of last iteration. Suppose \(a_i\) is left of/below \(a_j\). Then we check whether \(a_i\) has any predecessor in current \(G_v^*\) which is not below/left-of the reference-block (if the reference-block is no_def, we just ignore this requirement). If not, stops; otherwise, we pick up the one among them with the highest index in \(\Gamma_+\) \(a_k\). Let \(j = k\), goto ii again.

**Lemma 2** The constraint graph \(G_h^*\) and \(G_v^*\) output by algorithm NRCG is equivalent to the SP which it is built from in terms of topological constraints.

This is easy to prove following the line of thought in the proof of Lemma 1. The details are omitted here due to the limit of space.

**Lemma 3** Algorithm NRCG does not produce any redundant edge in either \(G_h^*\) or \(G_v^*\).

**Proof** Take the example in Fig 4. Suppose \(a_i = h\), \(a_j = c\) now and block \(a\) is block \(c\)’s left-predecessor in \(G_v^*\) which is not left-of (i.e. is below) block \(f\) with the largest index in \(\Gamma_-\). Suppose no redundant edge has been produced so far, then the edge \((a, h)\) is not redundant either, in either of the two cases:

(i) \(a\) is left of \(h\). If edge \((a, h)\) is redundant in \(G_h^*\), there must be another block \(g\) such that the path from \(a\) to \(g\) and the path from \(g\) to \(h\) already exist in \(G_h^*\). As the reference block \(f\) is the last block before changing direction again, all blocks visited after \(f\) (up till \(c\)) are below \(h\). So block \(g\) can not be any of them; \(g\) is not \(f\) either, for \(a\) is below \(f\). So \(g\) must have higher index in \(\Gamma_-\) than \(f\), and be either right of or above \(f\). \(g\) can not be right \(f\); otherwise \((f, h)\) is redundant. \(g\) can not be above \(f\) either, otherwise \(g\) is above \(a\) the same as \(f\). So block \(g\) can not exist, and edge \((a, h)\) is not redundant.

(ii) \(a\) is below \(h\). If edge \((a, h)\) is redundant in \(G_v^*\), there must be another block \(g\) such that the path from \(a\) to \(g\) and the path from \(g\) to \(h\) already exist in \(G_v^*\). \(g\) is not \(c\), for \(c\) is right of \(a\). So \(g\) must have higher index in \(\Gamma_-\) than \(c\), and be either right of or above \(c\). \(g\) can not be above \(c\), otherwise \((c, h)\) is redundant. \(g\) can not be right of \(c\) either, otherwise \(g\) is right of \(a\) the same as \(c\). So block \(g\) can not exist, and edge \((a, h)\) is not redundant.
**Lemma 4** Algorithm NRCG output $G_h^*$ and $G_v^*$ in time linear to $(|E_h| + |E_v|)$.

**Proof** The main operations in this algorithm are comparisons of two blocks’ indicis in $\Gamma$, to decide their topological relation. Each comparison belongs to either of the two classes: (1) It leads to an edge to be added to $G_h^*$ or $G_v^*$; (2) It is used in selecting a proper left/below-predecessor of a block to continue the process, but does not yield an edge in $G_h^*$ or $G_v^*$. The total number of comparisons in the 1st class is obviously the total number of edges in $G_h^*$ and $G_v^*$. It can be proved that the total number of comparisons in the 2nd class is no more than the total number of edges in $G_h^*$ and $G_v^*$, for there is an injection from the set of comparisons in the 2nd class to the set of edges in $G_h^*$ and $G_v^*$. The detailed proof is omitted here due to the limit of space.

Apart from reducing the evaluation time remarkably, the main significance of this approach lies in that it makes it very convenient for building the related-vertices-grouped constraint graph in the following rectilinear block packing algorithm and finding a bottleneck path in the following soft block packing algorithm (both of which are critical steps in the corresponding algorithms) and therefore serves as the basis of the whole unified rectilinear/soft block packing algorithm. For convenience we will refer to the non-redundant constraint graphs $G_h^*$ and $G_v^*$ as $G_h$ and $G_v$ in the following sections.

**IV. Efficient Arbitrary Rectilinear Block (Convex & Concave) Packing**

1. *Fundamentals of rectilinear block packing*

Up to the present the rectilinear block packing problem is dealt with mostly by partitioning each rectilinear block $A$ into a set of rectangle sub-blocks, representing them individually, and packing the mixture of these rectangle sub-blocks and original rectangle blocks by some existing rectangle packing algorithms. Fig 6(a) and Fig 6(b) give examples of horizontal partitioning and vertical partitioning, respectively. A representation of a rectilinear packing by SP is therefore a pair of permutations of all the unit block (rectangle sub-block or original rectangle block) names.

![Fig 6 H/V-partition of A Rectilinear Block](image)

Certain measures need to be taken to guarantee the relative positions of sub-blocks of a same rectilinear block so that this rectilinear block can keep its original shape after the packing process (a SP satisfying this need is called a feasible SP). A most notable measure is to apply the three necessary sufficient conditions of the feasibility of SP, as proposed in [1].

**Condition-1**: For any H-partitioned / V-partitioned block $A$, the permutation pair of $A$ equals the H-Pair / V-Pair of $A$.

**Condition-2**: Any two unit blocks $a_i, a_j \in A$ are not interrupted by a unit block $c \not\in A$.

**Condition-3**: Any two pairs of unit blocks $a_i, a_j \in A$ and $b_p, b_q \in B, A \neq B, (a_i, a_j)$ separates $(b_p, b_q)$ in the first or second sequence.
Where the unit block relations “interrupt” and “separate” are defined as:
- Given three unit blocks \( a_i, a_j \in A \) and \( c \notin A \), if \( c \) is between \( a_i \) and \( a_j \) in both sequences, e.g. \( (a_i, c, a_j, a_k) \), we call \( c \) interrupts \( a_i \) and \( a_j \).
- Given two pairs of unit blocks \( a_i, a_j \in A \) and \( b_p, b_q \in B, A \neq B \), if in the 1\textsuperscript{st}/2\textsuperscript{nd} sequence: \( a_i .. a_j .. b_p .. b_q \) or \( b_p .. b_q .. a_i .. a_j \), we \((a_i, a_j)\) and \((b_p, b_q)\) as separates of each other in the 1\textsuperscript{st}/2\textsuperscript{nd} sequence.

2. Packing rectilinear blocks based on a related-vertics-grouped constraint graph

In SP-based rectilinear block packing, as SP only defines the topological constraints among all the unit blocks (including both original rectangle blocks and rectangle sub-blocks of rectilinear blocks), the constraint graph built from SP doesn’t contain information about how the sub-blocks of a same rectilinear block should be placed relative to each other. Therefore the packing obtained from such constraint graph needs a post-process alignment to adjust the positions of the unit blocks such that the relative positions of each rectilinear block’s sub-blocks conform to its initial shape and the topological constraints among all unit blocks are preserved. Such separate compaction and alignment makes it difficult for shape optimization of floorplans containing rectilinear blocks. In the following we will show how to bypass this difficulty by a new method of rectilinear packing.

Therefore we call them related-vertics-grouped constraint graphs. The edges in \( G_h \) will be obtained in the following way:

For each edge \((v_p, v_q)\) in \( G_h \),

(i) If both \( v_p \) and \( v_q \) represent original rectangle blocks \( a \) and \( b \), we add an edge \((a, b)\) in \( G_h' \) and the weight remains the same;

(ii) If \( v_p \) represents a sub-block \( a_i \) of rectilinear block \( A \), \( v_q \) represents an original rectangle block \( b \), we add an edge \((A, b)\) in \( G_h' \) if it’s not present in \( G_h \) yet, and the weight is \( dx(a_i) + w(a_i) \); otherwise, we update the weight of the existing edge with \( \max(old\_weight, dx(a_i) + w(a_i)) \). Where \( dx(a) \) is the horizontal distance from the bottom-left corner of \( a \) to the bottom-left corner of the bounding box of \( A \) under current orientation of \( A \), \( w(a) \) is the width of \( a \);

(iii) If \( v_p \) represents an original rectangle block \( b \), \( v_q \) represents a sub-block \( a_i \) of rectilinear block \( A \), we add an edge \((b, A)\) in \( G_h' \) if it’s not present in \( G_h \) yet, and the weight is \( w(b) - dx(a_i) \); otherwise, we update the weight of the existing edge with \( \max(old\_weight, w(b) - dx(a_i)) \);

(iv) If \( v_p \) represents a sub-block \( a_i \) of rectilinear block \( A \), \( v_q \) represents a sub-block \( b_j \) of another rectilinear block \( B \), we add an edge \((A, B)\) in \( G_h' \) if it’s not present in \( G_h \) yet, and the weight is \( dx(a_i) + w(a_i) - dx(b_j) \); otherwise, we update the weight of the existing edge with \( \max(old\_weight, dx(a_i) + w(a_i) - dx(b_j)) \);

(v) If \( v_p \) and \( v_q \) represent two sub-blocks \( a_i \) and \( a_j \) of a same rectilinear block \( A \), we add an edge \((A, A)\) in \( G_h' \) if if it’s not present in \( G_h \) yet and the weight \( dx(a_i) + w(a_i) - dx(a_j) \) is not zero.

Then, we add an edge from the horizontal \( s_h \) source to every block in \( G_h' \) and the weight is zero. This is not significant in rectangle block packing, but it’s indispensable here. It is to prevent the coordinate of any block from being negative. Look at the example show in Fig 7. As \( b \) is left of \( a_2 \),
$b$ is left of $A$ in $G_h'$. Therefore $b$'s x-coordinate is calculated before $A$. As $b$'s x-coordinate is zero and $A$'s relative position to $b$ is negative, $A$'s x-coordinate is going to be negative without an edge $(s_h, A)$ in $G_h'$ with weight zero.

![Fig 7 Illustration of Why Edges Connected to Source is Needed](image-url)

The edges in $G_v'$ can be obtained similarly.

After $G_h'$ and $G_v'$ are drawn, we derive corresponding horizontal and vertical topological orders from them, and then we can get the location of each block (either rectangle or rectilinear) by applying the longest path algorithm to $G_h'$ and $G_v'$ in such orders.

**Lemma 5** The algorithm described above can achieve simultaneous compaction and alignment for rectilinear block packing.

**Proof** As we treat each rectilinear block as a whole, obviously the relative positions among sub-blocks of a same rectilinear block have been taken care of. Now let’s prove that $G_h'$ and $G_v'$ have preserved the topological constraints contained in $G_h$ and $G_v$. There must be no doubt about operation (i). Operations (ii), (iii) and (iv) are illustrated in Fig 8 (a), (b) and (c), respectively. In (a), there is an edge $(a_1, b)$ in $G_h$, which means $b$ must be right of $a_1$. Therefore the bottom-left corner of $b$ must be right of the bottom-left corner of $a_i$ by the amount of at least $w(a_1)$. This is equivalent to the constraint that the bottom-left corner of $b$ must be right of the bottom-left corner of the rectilinear block $A$ by the amount of at least $dx(a_1) + w(a_1)$, for the bottom-left corner of every sub-block $a_i$ to the bottom-left corner of the whole rectilinear block is always a constant $dx(a_i)$. In (b), there is an edge $(b, a_2)$ in $G_h$, which means $a_2$ must be right of $b$. Therefore the bottom-left corner of $a_2$ must be right of the bottom-left corner of $b$ by the amount of at least $w(b)$. This is equivalent to the constraint that the bottom-left corner of the rectilinear block $A$ must be “right of” the bottom-left corner of block $b$ by the amount of at least $w(b) - dx(a_2)$ for the reason as stated for Fig 8(a). Note that this value may be negative, which means rectilinear block $A$ can be left of block $b$ by the amount of at least $|w(b) - dx(a_2)|$. Combining the two facts together, we can easily defer the formula in (iv). And case (v) is similar to (iv). The operation in (v) is useful only for checking the feasibility of the corresponding SP. If the SP is feasible, no edge is added in (v), otherwise, a positive self-circle is added indicating the infeasibility of the SP.

And once we have this pair of related-verts-groups constraint graph $G_h'$ and $G_v'$, and the corresponding horizontal and vertical topological orders, we can apply the longest path algorithm to them just the same as in rectangle block packing, and get the x/y-coordinate of each rectilinear/rectangle block accordingly.

This constraint graph can also easily tell whether the corresponding sequence pair is feasible:

**Lemma 6** A sequence pair is feasible if and only if its corresponding related-verts-groups constraint graphs $G_h$ and $G_v$ contain no positive circle.
The detailed proof is omitted here due to the limit of space. This lemma is valid for the cases of both convex and concave rectilinear block packing. For convex rectilinear block packing, it is equivalent to the three necessary and sufficient conditions given by [1] as described in the previous sub-section. Condition-1 is satisfied if and only if either \( G_h' \) or \( G_v' \) contains no self-circle. Condition-2 and condition-3 are satisfied if and only if either \( G_h' \) or \( G_v' \) contains no circle among any two or more blocks.

This rectilinear block packing approach is very critical to the unified rectilinear/soft block packing system. Compared to the approach in [1], it achieves simultaneous compaction and alignment, and therefore solves the dilemma between post-alignment of rectilinear blocks and shape optimization of soft blocks. Compared to the approach in [3], first, it decreases the size of the problem by reducing the total number of vertices in the constraint graph instead of increasing the problem size by increasing the total number of edges in the constraint graph as done in [3]. Second, and more importantly, although [3] achieves simultaneous compaction and alignment too, it still represents the sub-blocks of a rectilinear block individually in the constraint graph; in contrast our approach represents the rectilinear block as a whole by one vertex and therefore the rectilinear block can be treated virtually as a rectangle block in the soft block packing. This makes it very easy to integrate rectilinear block packing with soft block packing.

V. Enhanced Soft Block Packing

1. Fundamentals of soft block packing

The soft block packing problem is to optimize packing topology as well as the block shapes to achieve minimum total packing area. The key issue is how to optimize the block shapes for a given topology. An attractive approach for this is proposed by [4], where the block shapes are gradually adjusted to reduce the overall height and overall width alternatively and monotonously for a given topology. This strategy is illustrated by Fig 9.

A greedy algorithm is applied to perform each step of overall height reduction and overall width reduction based on a metric called slack. For the SP structure, the horizontal slack of a block \( b \) can be given as:

\[
sl_h = l(s_h, t_h) - l(s_h, v_i) - l(v_i, t_h) - w(b)
\]

where \( s_h \) and \( t_h \) are the horizontal source and sink, respectively; and \( v_i \) is the vertex representing block \( b \) in \( G_h \). A block whose horizontal / vertical slack is zero is called \( h\text{-critical} \) / \( v\text{-critical} \) block. The horizontal maximal slack \( sl_h' \) of a block \( b \) can be obtained by increasing the height of \( b \) until it becomes \( v\text{-critical} \).
The following sufficient condition is used to meet the non-width-increase requirement:

\[ \sum_{b \in p_h} \Delta w_b \leq s^l (p_h) \]  

(1)

where the maximal slack of a horizontal path \( p \) is defined as:

\[ s^l (p) = \min_{b \in p} s^l_b \]

And according to this sufficient condition (1), the horizontal source-to-sink path to yield maximum overall height reduction without overall width increase (denoted as bottleneck path) can be found as a path with maximum \( s^l (p_h) \), where \( \text{num}(p_h) \) denotes the number of blocks on the path.

Then the shape of each block on the bottleneck path will be adjusted according to the following formula:

\[ \Delta w_b = s^l (p_h) \times \frac{w_i/b_i}{\sum_{b \in p_h} w_i/b_i} \]

(2)

The height of each block will be reduced by approximately equal amount and so does the overall height.

The monotonous height reduction and width reduction will be carried out alternatively until no further reduction can be made.

2. Our algorithm

Our soft block packing follows the basic strategy presented in [4]. Yet we make certain improvement and a very important enhancement as described in the following.

Firstly, as we observed that among all blocks on a horizontal path, only those v-critical blocks are critical to the reduction of the overall height, so we only adjust the shapes of those v-critical blocks and leave the non-v-critical blocks alone. Experiments prove that by doing so we are able to obtain a greater amount of height reduction in an iteration in most cases.

As the maximal slack of each v-critical block is the same as its slack, the concept of maximal slack is no longer needed here. Instead, as soft packing is usually under certain aspect ratio constraint, we introduce another metric feasible slack to reflect this constraint. The horizontal feasible slack of a block \( b \) is:

\[ s_{lf} (b) = \min (s_{lb}, w_{in}(b) - w(b)) \]
where \( w_m(b) \) is the maximum width of the block which is imposed by the minimum aspect ratio constraint; \( w(b) \) is the current width of the block \( b \).

Accordingly we substitute formula (1) in the above for the following sufficient condition of non-width-increase:

**Sufficient Condition 1**

\[
\sum_{b \in p_b} \Delta w_b \leq s_{f_j}(p_b)
\]

where the feasible slack of a horizontal path \( p \) is defined as:

\[
s_{f_j}(p) = \frac{\text{bis v-critical}}{\min_{b \in p} s_{f_j}(b)}
\]

Also accordingly we can find the bottleneck path as a path with maximum \( s_{f_j}(p_h) \), where \( \text{num}(p_h) \) denotes the number of all v-critical blocks on the path.

Then we can adjust the shape of each v-critical block on the bottleneck according to the following formula:

\[
\Delta w_b = s_{f_j}(p_h) \times \frac{w_b / h_b}{\sum_{b \in p_b} w_b / h_b}
\]

And we can carry out overall height reduction and overall width reduction alternatively until no further reduction can be made.

But something is missing here for both the initial algorithm and the improved algorithm. Look at the example in Fig 10 (Note that the sources and sinks are omitted here). It can be observed that a horizontal path (such as \( (e, d) \)) is not sure to intersect with a vertical path (such as \( (a, b, c) \)) on a block. So not every horizontal source-to-sink path contains a block on every vertical longest path. And therefore not every horizontal source-to-sink path is qualified as a horizontal bottleneck path.

We will find out a condition for such qualification in the following.

If a horizontal source-to-sink path \( p_h \) does not intersect with a vertical longest path \( p_v \) on any block, then there must be some edge (e.g. \( (e, d) \)) on \( p_h \) (e.g. \( (e, d) \)) crossing some edge (e.g. \( (b, c) \)) on \( p_v \) (e.g. \( (b, c) \)) as shown in the example. We will show that in such case there are only two possible combinations how vertices \( e \) and \( d \) is located relative to vertices \( b \) and \( c \).

**Lemma 7** If a horizontal edge \( (e, d) \) crosses a vertical edge \( (b, c) \), then \( e, d \) can only have two combinations of locations relative to \( b, c \): (i) \( e \) is in \( (7) \) and \( d \) is in \( (6) \); (ii) \( e \) is in \( (5) \) and \( d \) is in \( (8) \).

**Proof** Firstly, neither \( e \) nor \( d \) could be in region \( (1) \) or \( (2) \), otherwise \( (e, d) \) could not have crossed \( (b, c) \);

Secondly, neither \( e \) nor \( d \) could be in region \( (9) \), otherwise there should not have been an edge between \( b \) and \( c \) because any block in region \( (9) \) would make \( (b, c) \) a redundant edge.

Thirdly, if \( e \) is in region \( (3) \), \( e \) is left of \( b \) and \( c \), then \( d \) could only be in region \( (4), (6) \) or \( (8) \), for otherwise \( (e, d) \) could not have crossed \( (b, c) \). But in any of the three cases, \( d \) is transitively right of \( e \) through either \( b \) or \( c \) and therefore \( (e, d) \) would have become redundant and could not have existed. Therefore \( e \) could not be in region \( (3) \); similarly \( d \) could not be in region \( (4) \);

Fourthly, if \( e \) is in region \( (5) \), then \( d \) could not be in region \( (6) \), otherwise \( d \) is transitively right of \( e \) through \( b \) and therefore \( (e, d) \) could not have existed; similarly if \( e \) is in region \( (7) \), then \( d \) could not be in region \( (8) \).
For case (i) in lemma 7, $e$ is above $b$ and $d$ is below $c$. As $(b, c)$ is a v-critical edge, $b$’s lower boundary must lie on the same horizontal line as $c$’s upper boundary. Therefore $y(e_{\text{lower}}) \geq y(b_{\text{upper}}) = y(c_{\text{lower}}) \geq y(d_{\text{upper}})$, and $e$ and $d$ can not have overlap in y-direction, as in the example below. (Note that $e$ and $d$ is not considered as having overlap in y-direction if $e$’s lower boundary lies on the same horizontal line as $d$’s upper boundary). Similarly, and $e$ and $d$ can not have overlap in y-direction for case (ii).

Thereby we can give a sufficient condition of a horizontal path $p_h$ intersecting every vertical longest path on a block:

**Sufficient Condition 2** Every two adjacent blocks on a horizontal path $p_h$ must have overlap in y-direction.

Based on this sufficient condition, we can modify the algorithm described above accordingly. First we give two definitions: If there is an edge $(a, b)$ in $G_{h}$, we say that $a$ is $b$’s topological predecessor; if $a$ and $b$ furthermore have overlap in y-direction, then we say that $a$ is also $b$’s geometrical predecessor. Then we make the following modifications: (1) Not only any block with no topological predecessor/successor, but also any block with no geometrical predecessor/successor, could be the beginning/end of a bottleneck path. (2) For every block $b$ except the beginning of a horizontal source-to-sink path, we calculate its $sl(v_i)$ and $num(v_i)$ only based on its geometrical predecessors, where $v_i$ is the vertex representing $b$ in the constraint graph, $sl(v_i)$ is the feasible slack of the piece of path on the whole path from the beginning up to $v_i$.

### VI. Searching The Solution Space by Global Moves

Based on the three necessary and sufficient conditions as described in section IV-1, [1] defines three kinds of stochastic moves and corresponding adaption procedure.

**Rotation**: randomly pick up a macro block (rectilinear or rectangle) $A = \{a_1, a_2 \ldots a_m\}$ and rotate it by 90 in the clockwise direction. The sequence pair is accordingly changed by switching unit block $a_i$ with $a_{m+1-i}$, $i \in [1, n]$, in $\Gamma_+$ (when changing $A$ from an H-partitioned orientation to a V-partitioned orientation) or in $\Gamma_-$ (when it is the other way round).

**$\Gamma_+$-mutation**: randomly pick up two adjacent unit blocks $a \in A$ and $b \in B$ ($A \neq B$) in $\Gamma_+$ and exchange them.

**$\Gamma_-$-mutation**: randomly pick up two adjacent unit blocks $a \in A$ and $b \in B$ ($A \neq B$) in $\Gamma_-$ and exchange them.

These moves and adaption procedure defined by [1] are attractive for adoption in the case concave rectilinear blocks are not present, due to its quick and easy operation. Therefore with the absence of
concave rectilinear blocks we would make our stochastic search by following the same line as [1]. In [1], the $\Gamma_+$-mutation and $\Gamma_-$-mutation are restricted to exchanging of blocks between two adjacent blocks in either $\Gamma_+$ or $\Gamma_-$. In our approach, we extend $\Gamma_+$-mutation and $\Gamma_-$-mutation to exchanging of blocks between two blocks in either $\Gamma_+$ or $\Gamma_-$ with any distance under certain restrictions.

$\Gamma_+$-mutation: randomly pick up two unit blocks $a \in A$ and $b \in B (A \neq B)$ in $\Gamma_+$ such that no other unit blocks belonging to the either A or B are in between. Exchange $a$ and $b$ in $\Gamma_+$.

$\Gamma_-$-mutation: randomly pick up two unit blocks $a \in A$ and $b \in B (A \neq B)$ in $\Gamma_-$ such that no other unit blocks belonging to the either A or B are in between. Exchange $a$ and $b$ in $\Gamma_-$.

Like in [1], the $\Gamma_+$-mutation/$\Gamma_-$-mutation may result in the violation of Condition-2 or Condition-3 defined by [1] under certain circumstances. So each $\Gamma_+$-mutation / $\Gamma_-$-mutation will be followed by a check of feasibility of the resultant SP and an adaption in the case of infeasibility. The details are omitted here due to the limit of space.

VII. Assembly Everything Together

Because all the topological constraints and relative position constraints have been nicely captured in the related-vertices-grouped constraint graph $G_h$ and $G_v$, and every rectilinear block is represented as one vertex, just as if it was a rectangle block. So carrying out soft block packing on floorplans with rectilinear blocks will not be any more complicated than on those without rectilinear blocks. We only need to set the rectilinear blocks’ horizontal/vertical feasible slack to zero to indicate they are hard blocks and can then perform the soft block packing as if no rectilinear block is present.

The whole algorithm uses simulated annealing mechanism to search for an optimal topology in the feasible solution space of sequence pairs of all the unit blocks. The key operations at each step of the simulated annealing is making a move and evaluating the resultant sequence pair.

Making a move including the following steps:
1. Select a move and make it;
2. Check the feasibility of the resultant SP if the move is $\Gamma_+$-mutation or $\Gamma_-$-mutation;
3. Adapt the SP if it’s infeasible.

Evaluating a SP including the following steps:
1. Construct the non-redundant constraint graph $G_h$ and $G_v$ which include all unit blocks for this SP;
2. Construct the corresponding sub-blocks-grouped constraint graph $G_h'$ and $G_v'$;
3. Perform soft block packing based on $G_h$ and $G_v$;
4. Calculate the cost of the packing output by (3).

VIII. Experiment Results

We have implemented the algorithm in C++(STL) with a Java interface and tested in on a SUN ULTRA-450 workstation. Fig 3.11(a), (b), (c) shows the packing results of 3 randomly generated sets of blocks. Set (a) is a packing of 10 blocks, 2 of them are rectilinear blocks. Either Set (a) or (c) is a packing of 20 blocks, 4 of them are rectilinear blocks. All the rectangle blocks are soft blocks whose shapes are optimized during the packing process. It takes no more than several minutes to get such a packing for any of the three cases.
IX. Concluding Remarks

In this paper, we propose a new technique of evaluating sequence pair; We present a new method of simultaneous compaction and alignment for rectilinear block packing; We make a key observation for soft block packing problem and give a enhanced soft block packing algorithm; And most importantly, we unifies the rectilinear block packing and soft block packing into a whole system for the first time; Also, we give an extension of stochastic moves to search the solution space more effectively.

Reference