Optimal Folding Of Bit Sliced Stacks+

Doowon Paik Sartaj Sahni

University of Minnesota University of Florida

Abstract

We develop fast polynomial time algorithms to optimally fold stacked bit sliced architectures to minimize area subject to height or width constraints. These algorithms may also be applied to folding problems that arise in standard cell and sea-of-gates designs.

KEYWORDS and PHRASES

Stacked bit sliced architectures, folding, area

+ Research supported, in part, by the National Science Foundation under grant MIP 86-17374.

1 Introduction

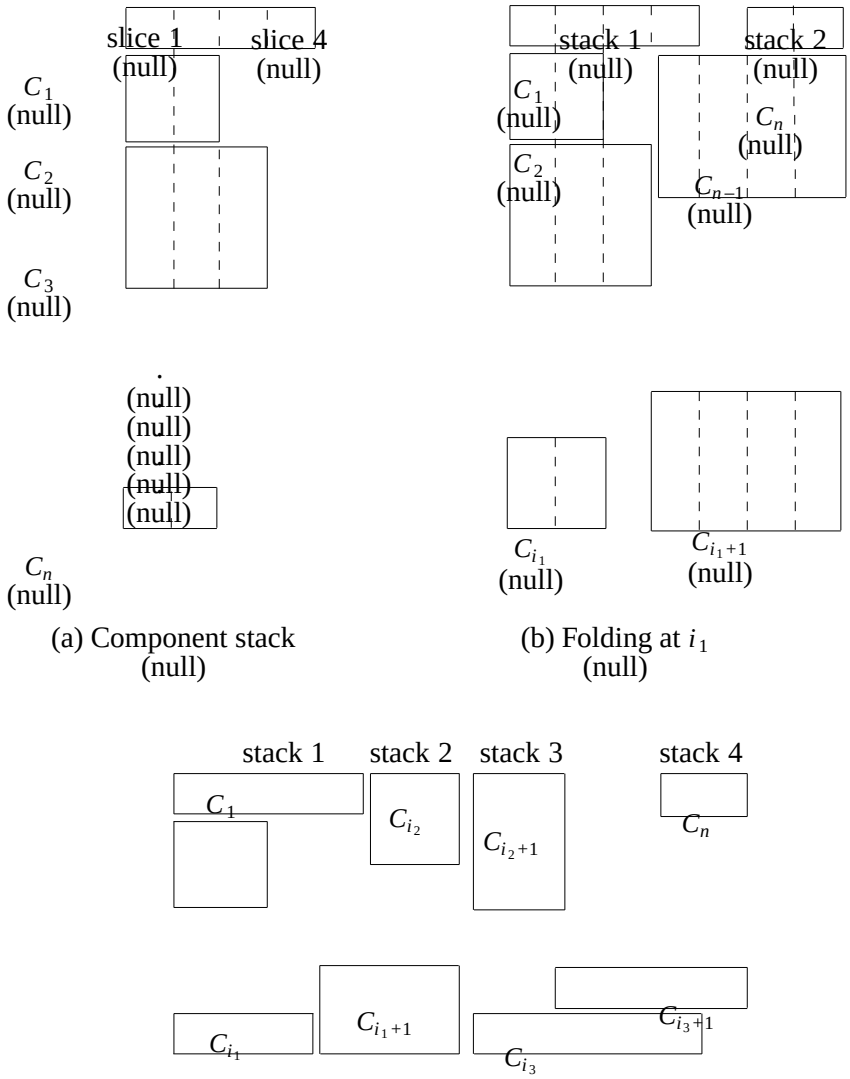
A stack of bit sliced components ([LARM90] and [WU90]) consists of n components of varying height and width with their left ends vertically aligned as in Figure 1(a). The intra component (i.e., inter slice but local to a component) routing is done on metal layer 1 while the inter component (i.e., intra slice but across components) routing is done on metal layer 2.

A component stack may be folded at component i_1 by rotating components i_1+1, ... , i_n by 180 ° so that their aligned ends are now on the right and the component order is i_1+1, ... , i_n bottom to top (Figure 1(b)). As a result of this, the slices of components i_1+1, ... , i_n are in the order slice 1, slice 2, ... , right to left. Folding at i_1 creats two components stacks. One is left aligned (i.e., components 1, ..., i_1) and the other is right aligned (components i_1+1, ... , i_n). By folding at i_1, i_2, and i_3, a four stack arrangement as in Figure 1(c) is obtained.

When a component stack is folded as in Figure 1(b), it becomes necessary for the available vertical tracks in a physical slice to be able to carry the routes for two logical slices. For example, in Figure 1(b), slice 4 of component C_1 and slice 4 of component C_{n-1} occupy the same physical chip space. We assume that the width of a slice is sufficient for this. Further, when a stack is folded at component i_1 we may need space at the ends of the two created stacks to complete the routes between components i_1 and i_1+1. If this can be done in a third layer, then no such routing space is needed. If not, additional space proportional to the number of wires between i_1 and i_1+1 must be reserved (Figure 2). We consider both of these cases in this paper.

The bit sliced model defined above was introduced by Larmore, Gajski, and Wu [LARM90] for the compilation of arbitrary net lists into layout for CMOS technology. They studied various folding strategies under the assumtion that components may be ordered and the stack can be folded only once (so, only a two stack configuration as in Figure 1(b) is possible). These folding schemes begin by reordering the components by width (i.e., $w_i \geq w_{i+1}$, $1 \leq i < n$ where w_i is the width, in number of slices, of component i). In [Wu90], Wu and Gajski report on the application of their two stack folding model to real circuits.

In this paper, we place no restriction on the number of folding points $i_1, i_2, \ldots, i_k$. Additionally, we do not permit component reordering. This restriction is realistic as the component stack is usually ordered so as to minimize inter component routing requirements and optimize performance. We consider both the situation when extra routing space at the stack ends is and is not



(a) Component stack

(b) Folding at i_1

(c) Folded into 4 stacks

Figure 1: Stack of bit sliced components.

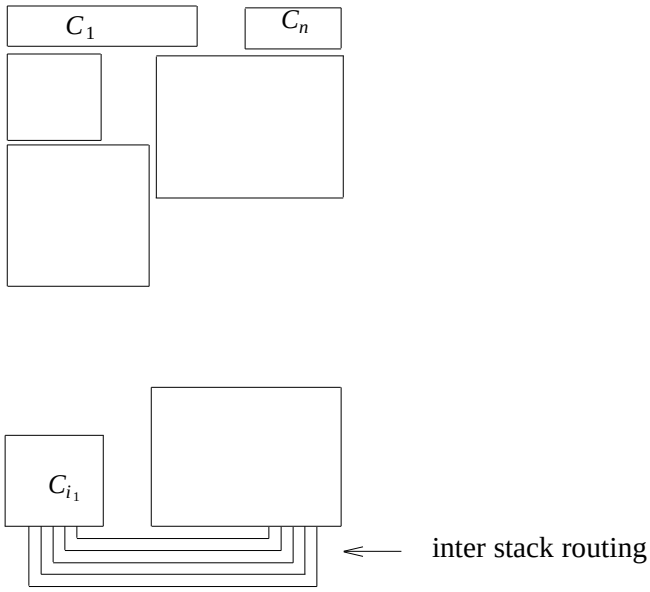


Figure 2: Routing space reserved.

needed to accomodate routes between components at the ends of adjacent stacks. Note that our model can also be used for the folding step of placement algorithms for standard cell and sea-of-gates designs [SHRA88, SHRA90]. In this, the modules to be placed have been ordered by some criterion and are then folded into the layout area so as to minimize area. In the case of standard cell designs, all modules have the same width while the case of sea-of-gates designs module widths and heights vary from module to module.

The objective of folding is to obtain a minimum area layout subject to a height or width constraint. We consider both types of constraints here. Note that if the height (width) is constrained to be h (w), then the area is minimized by minimizing the width (height). Note also that if the height (width) is constrained to be h (w), it is enough to fold the component stack so that the height (width) of the bounding rectangle is less than or equal to h (w). However, if the height (width) is $\leq h$ (w), the physical chip space allocated will have height (width) h (w). So, area is

minimized by minimizing the width (height).

In Section 2, we consider folding under the assumtion that all module widths are the same (module heights may vary). The case when module widths vary but module heights are the same is considered in Section 3. The general case of variable module widths and heights is considered in Section 4. In Section 5, we consider the case when no overlap amongst the slices of different stacks is permitted. This is referred to as folding without nesting.

2 Components With Equal Widths

Let h_i be the height of the i'th component. We may assume that all components have a width of one. If the stack is folded at components $i_1, i_2, \ldots, i_{k-1}$, then the width of the layout is k and the height, h, is the height of the tallest of the k stacks (Figure 3).

2.1 No Routing Area At Stack Ends

2.1.1 Height Of Folded Layout Is Fixed

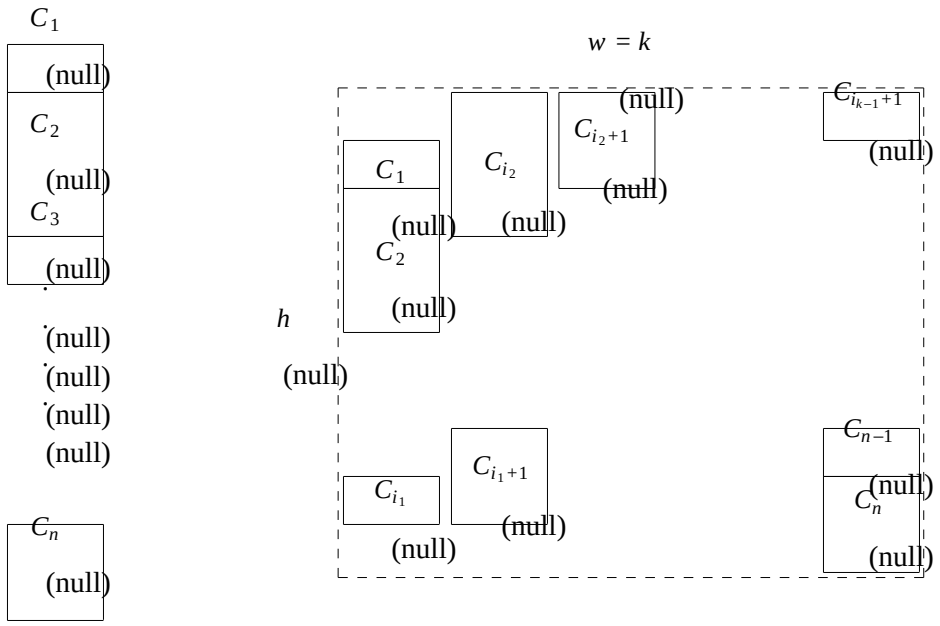
When the height is constrained to be at most h, the minimum width layout is obtained by first selecting i_1 to be the largest j such that $\sum_{i=1}^{j} h_i \le h$. Next i_2 is set to be the largest j such that $\sum_{i=i_1+1}^{j} h_i \le h$. Continuing in this way, $i_3, i_4, \ldots$ can be obtained. The process stops when $i_{k-1} = n$. The number of stacks is k. The optimality of this strategy is easily established and its complexity is readily seen to be $O(n)$.

2.1.2 Width Of Folded Layout Is Fixed

If the width of the layout is constrained to be at most k, the minimum height layout can be obtained by using a dynamic programming technique. Let $H(i, w)$ be the height of the rectangle with width w and minimum height into which components $C_i, C_{i+1}, \ldots, C_n$ can be folded. It is easy to see that the following equalities are true:

$$H(i, 1) = \sum_{j=i}^{n} h_j$$

$$H(n, w) = h_n \quad \text{for all } w,\ w \ge 1$$



(a) Unfolded placement

(b) Folded placement and enclosing rectangle

Figure 3: A k stack folding of equal width components.

$$H(n+1, w) = 0 \quad \text{for all } w,\ w \geq 1$$

Let $h(i, j) = \sum_{t=i}^{j} h_t$. A recurrence for $H(i, w)$ in terms of the heights of smaller width foldings is obtained by observing that if $C_i, C_{i+1}, \ldots, C_n$ is first folded at C_j, then the height of the enclosing width w rectangle is $\max\{ h(i, j), H(j+1, w-1) \}$. So,

$$H(i, w) = \min_{i \leq j \leq n} [\ \max\{ h(i, j),\ H(j+1, w-1) \}\ ], \quad w > 1. \tag{1}$$

The minimum height folding of $C_1, C_2, \ldots, C_n$ constrained to have width at most k is

$H(1, k)$.

Using Equation (1) and the known values of $H(i, 1)$, $1 \le i \le n$, and $H(n+1, w)$, $H(i, 2)$ can be obtained. From the $H(i, 2)$'s and Equation (1) we can obtain the $H(i, 3)$'s. Proceeding in this way, the $H(i, k)$'s and hence $H(1, k)$ can be obtained. A straightforward application of (1) to compute each $H(i, w)$ will take $O(n-i+1)$ time. The total complexity to compute all $H(i, w)$'s, $1 \le i \le n$, $1 \le w \le k$ will be $O(n^2k)$. We can reduce this to $O(nk)$ by using the following results.

Lemma 1: Let j_i be a value of j, $i \le j \le n$ that minimizes max $\{ h(i, j), H(j+1, w-1) \}$ (see Equation (1)). For this value, max $\{ h(i, j_i), H(j_i+1, w-1) \} \le H(j_i, w-1)$.

Proof: Suppose $j_i = i$. Then, since $h(i, i) \le H(i, w-1)$ and $H(i+1, w-1) \le H(i, w-1)$, max $\{ h(i, i), H(i+1, w-1) \} \le H(i, w-1)$. So the lemma 1 is true for $j_i = i$. Suppose $j_i > i$. If max $\{ h(i, j_i), H(j_i+1, w-1) \} = H(j_i+1, w-1)$, then the lemma follows from the observation that $H(j_i+1, w-1) \le H(j_i, w-1)$. So, consider the case $h(i, j_i) > H(j_i+1, w-1)$. If the lemma is not true, then $h(i, j_i) > H(j_i, w-1)$. From this and the observation that $h(i, j_i-1) < h(i, j_i)$, it follows that max $\{ h(i, j_i-1), H(j_i, w-1) \} < h(i, j_i)$. This contradicts the assumption that j_i is a value of j that minimizes max $\{ h(i, j), H(j+1, w-1) \}$. □

Lemma 2: Let j_i be the largest value of j, $i \le j \le n$ that minimizes max $\{ h(i, j), H(j+1, w-1) \}$. Let j_{i+1} be similarly defined. Then $j_i \le j_{i+1}$, $1 \le i < n$.

Proof: max $\{ h(i+1, j_i), H(j_i+1, w-1) \}$

$\le$ max $\{ h(i, j_i), H(j_i+1, w-1) \}$ (as $h(i, j_i) > h(i+1, j_i)$)

$\le H(j_i, w-1)$ (Lemma 1)

$\le H(p+1, w-1)$ $0 \le p < j_i$ (as $H(a, w-1) \ge H(a+1, w-1)$ for all a)

$\le$ max $\{ h(i+1, p), H(p+1, w-1) \}$, $0 \le p < j_i$

So, max $\{ h(i+1, j), H(j+1, w-1) \}$ is minimized for a value of j that is $\ge j_i$. Hence, $j_i \le j_{i+1}$. □

Lemma 3: If j_i (as in Lemma 2) is known, then j_{i+1} (Lemma 2) can be found in $O(j_{i+1}-j_i+1)$ time if the $h(i+1, j)$'s and $H(j, w-1)$'s are known.

Proof: Since $h(i+1, j)$ is an increasing sequence (i.e., $h(i+1, j) < h(i+1, j+1)$ for all $j < n$) and $H(j+1, w-1)$ is a nonincreasing sequence, $f(i+1, j) = \max\{ h(i+1, j), H(j+1, w-1) \}$ is a bitonic sequence. I.e., the following is true for some q, $i < q \leq n$:

$$f(i+1, j) \geq f(i+1, j+1) \geq \ldots \geq f(i+1, q) \leq f(i+1, q+1) \leq \ldots \leq f(i+1, n)$$

Hence, j_{i+1} is the least j for which $f(i+1, j) < f(i+1, j+1)$ (in case there is no such j, $j = n$). Since $j_{i+1} \geq j_i$ (Lemma 2), the search for j_{i+1} can be confined to the interval $[j_i, n]$. So, we need to check if $f(i+1, j) < f(i+1, j+1)$ only for j values in the range $[j_i, j_{i+1}]$. This takes $O(j_{i+1}-j_i+1)$ time under the condition that $h(i+1, j)$'s and $H(j, w-1)$'s are known. □

Theorem 1: $H(i, w)$, $1 \leq i \leq n$, $1 \leq w \leq k$ can be computed in $O(nk)$ time.

Proof: The computation is done first for $w = 1$, then $w = 2$, $w = 3$, ..., $w = k$. For $w = 1$, $H(i, w)$, $1 \leq i \leq n$, is obtained in $O(n)$ time as $H(i, 1) = \sum_{j=i}^{n} h_j$. For $w > 1$, the computation is done in the order $i = 1, 2, \ldots, n$. For any i, the first value of h that is needed is $h(i, j_{i-1})$. This is just $h(i-1, j_{i-1}) - h_{i-1}$. So, $h(i, j_{i-1})$ can be obtained in $O(1)$ time as the value $h(i-1, j_{i-1})$ was computed during the computation of j_{i-1}. $h(i, j_{i-1}+r)$ can be obtained from $h(i, j_{i-1}+r-1)$ by adding h_r. So each of the needed $h(i, s)$ values can be obtained in $O(1)$ time. The last $h(i, s)$ value computed is $h(i, j_i)$ and is used to obtain $h(i+1, j_i)$. From Lemma 3, we know that a total of $j_i-j_{i-1}+1$ $h(i, s)$ values are needed and that the time to compute $H(i, w)$ is proportional to $j_i-j_{i-1}+1$. Further, the time to get $H(i, w)$ for all i, $1 \leq i \leq n$ is proportional to $\sum_{i=1}^{n} (j_i-j_{i-1}+1) = O(n)$. Hence, $H(i, w)$, $1 \leq i \leq n$, $1 \leq w \leq k$ can be computed in $O(nk)$ time. □

A bounding rectangle of dimension $h \times w$ is said to be a dominating rectangle iff the components $C_1, C_2, \ldots, C_n$ can be folded into it and there is no $H \times W$ rectangle, $H \leq h$, $W \leq w$, $H*W < h*w$, such that $C_1, C_2, \ldots, C_n$ can be folded into the $H \times W$ rectangle. The dominating rectangles can be found in $O(n^2)$ time by computing $H(1, w)$, $1 \leq w \leq n$ and eliminating from these n rectangles, those that are dominated by a rectangle in this set.

2.2 Routing Area At Stack Ends

Let r_i, $2 \leq i \leq n$ denote the height of the routing space needed to route the connections between C_{i-1} and C_i under the assumption that C_{i-1} and C_i are in different stacks (so, they must be at the same end of two adjacent stacks). Let $r_1 = r_{n+1} = 0$. Figure 4 shows the situation when C_{i-1} and C_i are at the bottom end. Note that r_i depends on the number of interconnects that pass from C_{i-1} to C_i but not on the relative positioning of C_{i-1} and C_i at the bottom of their respective stacks. Further, note that if components C_{i_1}, ... , C_{i_2} form one stack, then the height needed to accomodate these components and the inter stack routing to C_{i_1-1} and C_{i_2+1} is $\sum_{j=i_1}^{i_2} h_j + r_{i_1} + r_{i_2+1}$.

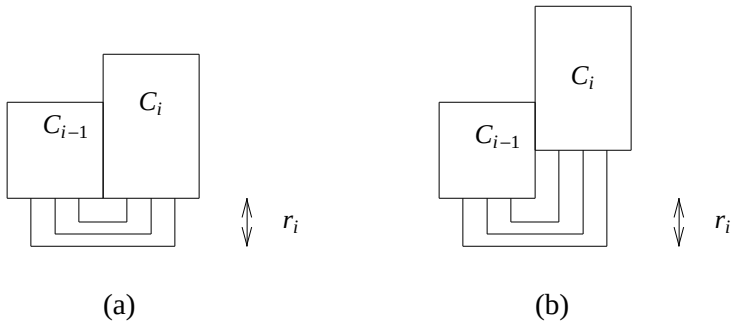


Figure 4: Inter stack routing height requirement.

2.2.1 Height Of Folded Layout Is Fixed

Let $W(j)$ be the width of a minimum width folding for C_j, ... , C_n. This folding is required to leave r_j space in the first stack to complete the routes between C_{j-1} and C_j. Note that it is possible that $r_j + h_j > h$ for some j. In this case $W(j) = \infty$ as it is not possible fold C_j, ... , C_n as desired. Whenever a height h folding is not possible, $W(j) = \infty$.

Let $W(n+1) = 0$. We see that

$$W(i) = 1 + \min \{ \{ W(j+1) \mid i \le j \le n, r_i + \sum_{q=i}^{j} h_q + r_{j+1} \le h \}, \infty \}.$$

The above recurrence can be solved for $W(1)$ in $O(n^2)$ time. The actual folding with width $W(1)$ can also be obtained in this much time.

2.2.2 Width Of Folded Layout Is Fixed

Let $H'(i, k)$ be the height of a minimum height rectangle of width k into which $C_i, C_{i+1}, \ldots, C_n$ can be folded. This folding requires that there be enough space at the ends of each of the k stacks to complete the inter stack routing. In particular, there must be r_i units of space (i.e., height) between the nearest rectangle boundary (top or bottom) and the end of C_i. This space is needed to complete the routing to C_{i-1}. Figure 5 illustrates this.

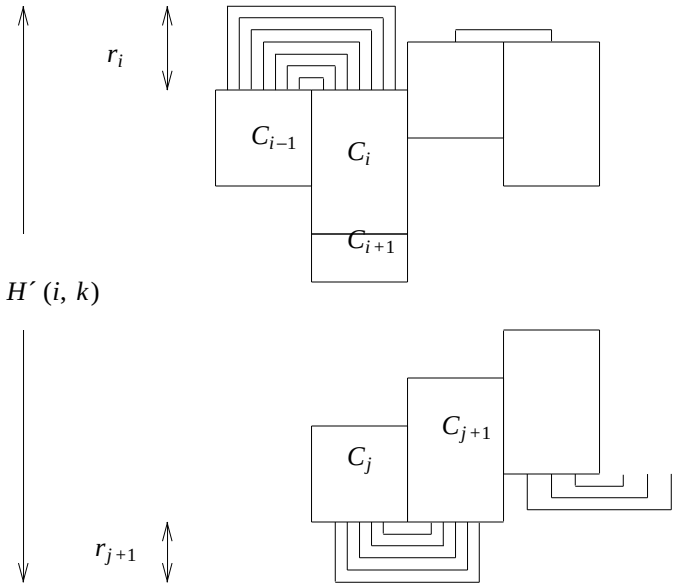


Figure 5: $H'(i, k)$

One may readily verify the correctness of the following equalities:

$$H'(i,\ 1) = h(i,\ n)\ +\ r_{i,} \qquad 1 \le i \le n$$

$$H'(n,\ w) = h_n\ +\ r_{n,} \qquad w \ge 1 \tag{2}$$

$$H'(i,\ w) = \min_{i \le j < n} [\ \max\ \{\ h(i,j) + r_i + r_{j+1}, \quad H'(j+1, w-1)\ \}\ ].$$

$H'(1, k)$ is the height of the minimum height rectangle with width k into which $C_1, \ldots, C_n$ can be folded and in which the inter stack routing can be completed. $H'(1, k)$ can be computed from (2) in $O(n^2k)$ time by computing $H'(i, w)$ for w in the order $w = 2, 3, \ldots, k$. The $O(nk)$ scheme of Section 2.1 does not generalize to (2) as $H'(i, w)$ is no longer monotone in i.

3 Components With Equal Heights

3.1 No Routing Area At Stack Ends

3.1.1 Height Of Folded Layout Is Fixed

First, consider the case when the height, h, of the folded layout is fixed and we wish to minimize the layout width. Let $W(i, h)$ be the width of a minimum width height h folded layout for the equal height components $C_i, \ldots, C_n$. Consider any folded layout for $C_i, \ldots, C_n$. This consists of a height h layout for $C_i, \ldots, C_j$ (for some j, $i \le j \le n$) which is folded at at most one position (Figure 6) followed by a folded layout for $C_{j+1}, \ldots, C_n$. Note that when $j = n$, there is only the layout for $C_i, \ldots, C_j$.

Let $S(i, j, h)$ be the width of a minimum width height h folded layout for $C_i, \ldots, C_j$ under the restriction that there is at most one folding position. Then it follows that

$$W(i,h) = \min_{i \le j \le n}\ \{\ S(i,\ j,\ h) + W(j+1,\ h)\} \tag{3}$$

and $W(n+1, h) = 0$.

Let w_i be the width of C_i, $1 \le i \le n$. We easily see that $W(n, h) = w_n$, for $h \ge 1$ (we assume that the height of each component C_i is 1). Since at most h components can be placed on a stack of height h and since $S(i, j, h)$ can contain at most two stacks, it follows that (3) may be rewritten as:

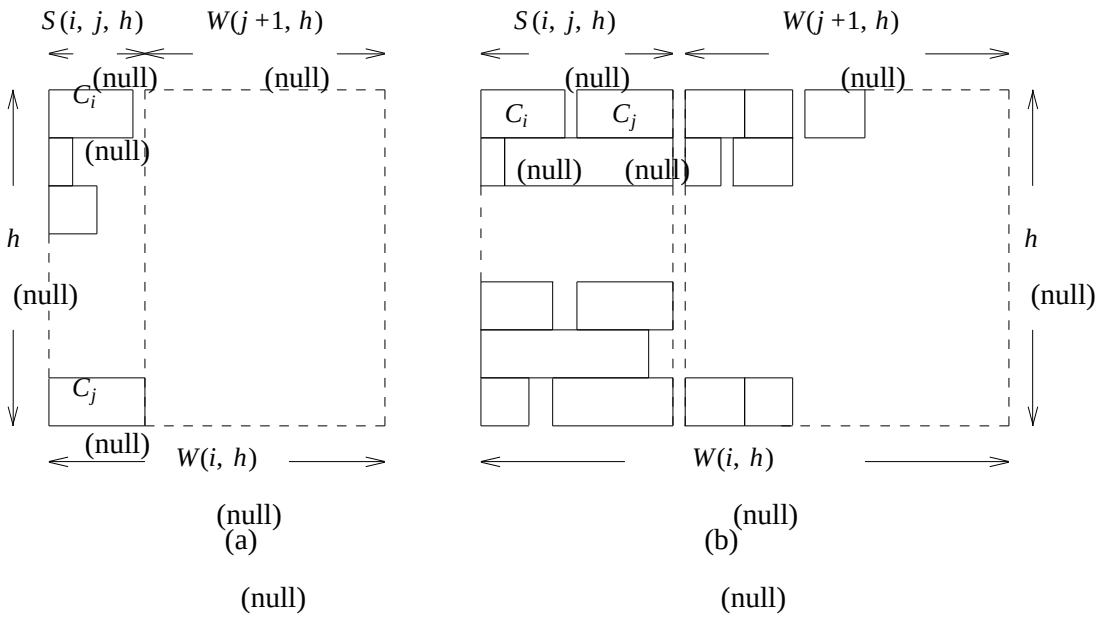


Figure 6: Possible folded layouts for C_i, ..., C_n.

$$W(i,\ h)\ =\ \min_{i \le j \le \min\ \{\ n,\ i+2h-1\ \}} \{\ S(i,\ j,\ h) + W(j+1,\ h)\ \} \qquad (3')$$

The minimum width height h folding we seek has width $W(1, h)$. To compute this width, we need to know $S(i, j, h)$ for $1 \le i \le n$ and $i \le j \le \min\ \{n, i + 2h - 1\}$. So we shall proceed to describe how $S(i, j, h)$ may be computed. With a little more book keeping, the layout corresponding to $S(i, j, h)$ may also be obtained.

To compute $S(i, j, h)$, we use the solution to a related problem P1 in which we are given two sets $L = \{L_1, \ldots, L_n\}$ and $R = \{R_1, \ldots, R_m\}$ of equal height components and a rectangle of width w. The width of L_i is wl_i and that of R_i is wr_i. An example is given in Figure 7(a). The width w rectangle is divided into buckets that are of unit height and width w as in Figure 7(b). The components of L and R are to be assigned to buckets such that:

(a) the fewest number of buckets are used

(b) each bucket contains at most one L_i and one R_j

(c) if a bucket contains L_i and R_j, then $wl_i + wr_j \le w$

(d) the order in which the L_i's (R_j's) are assigned to buckets is $L_1, L_2, \ldots, L_n$ ($R_1, R_2, \ldots, R_m$) bottom to top.

In order to assure a feasible solution, we require $w \ge \max \{ wl_1, \ldots, wl_n, wr_1, \ldots, wr_m \}$. Figure 7(c) shows a possible assignment of the L's and R's of Figure 7(a) into 4 buckets.

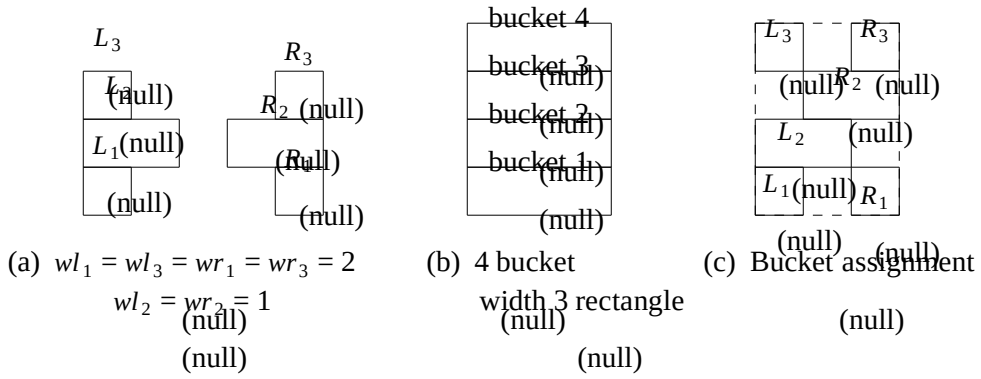


Figure 7: An example for P1.

Let $B(i, j, w)$ be the smallest number of buckets needed for $L_1, \ldots, L_i$ and $R_1, \ldots, R_j$. The number of buckets used in any solution of P1 is $B(n, m, w)$. It is easy to see that the following are true:

$$B(i, 0, w) = i, \quad 1 \le i \le n$$

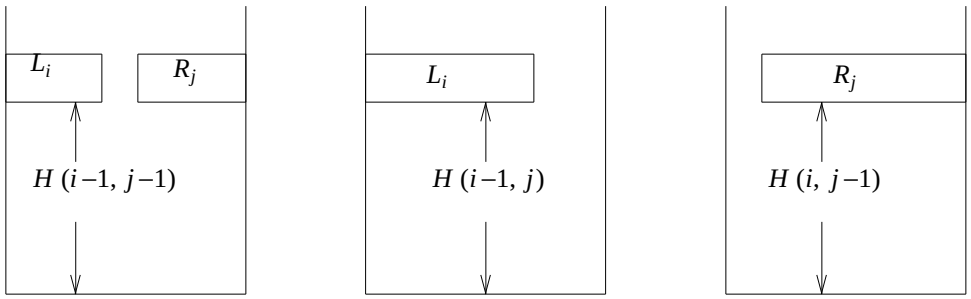
$$B(0, j, w) = j, \quad 1 \le j \le m$$

$$B(i, j, w) \ge B(i, j-1, w), \quad 1 \le i \le n, 1 \le j \le m$$

$$B(i, j, w) \ge B(i-1, j, w), \quad 1 \le i \le n, 1 \le j \le m$$

Furthermore, for an arbitrary $B(i, j, w)$, $i \geq 1$, $j \geq 1$, we see that if $wl_i + wr_j \leq w$, then there is no advantage to not putting L_i and R_j into the same bucket (see Figure 8(a)). Hence, $B(i, j, w) = 1 + B(i-1, j-1, w)$. If $wl_i + wr_j > w$, then L_i and R_j can not be placed in the same bucket. Because of the ordering requirement (d), we have two possibilities. In one, L_i occupies the highest used bucket. This bucket cannot be shared with any R_s as R_j must occupy the highest bucket occupied by any of $R_1, \ldots, R_j$ and R_j cannot fit into the same bucket as L_i. So, $B(i, j, w) = 1 + B(i-1, j, w)$ (Figure 8(b)). In the second case, R_j occupies the highest used bucket and the remaining components occupy lower buckets (Figure 8(c)). Now, $B(i, j, w) = 1 + B(i, j-1, w)$. Combining these observations together, we get:

$$B(i, j, w) = \begin{cases} 1 + B(i-1, j-1, w), & wl_i + wr_j \leq w \\ 1 + \min\{B(i-1, j, w),\ B(i, j-1, w)\}, & wl_i + wr_j > w \end{cases} \quad (4)$$



(a) $wl + wr \leq w$ (b) $wl + wr > w$ (c) $wl + wr > w$

Figure 8: Cases for placement of L_i and R_j.

Equation (4) together with the values of $B(i, 0, w)$, $1 \leq i \leq n$ and $B(0, j, w)$, $1 \leq j \leq m$, can be used to compute all $B(i, j, w)$ values, $1 \leq i \leq n$, $1 \leq j \leq m$ by computing these first for all i, j such that $i + j = 2$, then $i + j = 3$, ..., and finally $i + j = n + m$. The time needed to obtain $B(n, m, w)$ is therefore $O(nm)$. The actual assignment of the L_i's and R_j's to the buckets can be obtained by

recording which of the three possibilities of (3) results in each $B(i, j, w)$ value and then performing a traceback (see Horowitz and Sahni [HORO76] for a discussion of dynamic programming and associated tracebacks).

Let $T(i, j, k, w)$ be the minimum number of width w buckets needed for the problem P1 instance defined by $L = \{C_i, \ldots, C_k\}$, $R = \{C_j, \ldots, C_{k+1}\}$, $i \le k \le j$. Note that $T(i, j, k, w)$ gives the height of the minimum height width w rectangle into which $C_i, \ldots, C_j$ will fit when folded at C_k. Let $T(i, j, w) = \min_{i \le k \le j} \{ T(i, j, k, w) \}$. $T(i, j, w)$ is the height of a minimum height width w rectangle into which $C_i, \ldots, C_j$ will fit when the component stack $C_i, \ldots, C_j$ is folded at at most one component (recall that folding at $k = j$ corresponds to no folding). $T(i, j, w)$ can be obtained by first computing $T(i, j, k, w)$ using P1 as described above. Since each $T(i, j, k, w)$ can be computed in $O((k-i+1)(j-k))$ time, $T(i, j, w)$ can be computed in $O(\sum_{i \le k \le j} (k-i+1)(j-k)) = O((j-i)^3)$ time.

A faster way to obtain $T(i, j, w)$ is to solve the P1 instance defined by $L = \{L_1, L_2, \ldots, L_{j-i+1}\}$, $R = \{R_1, R_2, \ldots, R_{j-i+1}\}$, $wl_s = w_{j-s+1}$, $wr_s = w_{i+s-1}$, $1 \le s \le j-i+1$ where w_t is the width of component C_t. Let B be the $(j-i+1) \times (j-i+1)$ minimum bucket matrix computed using (4). $T(i, j, k, w) = B(j-k, k-i+1, w)$. To see this, consider Figure 9(a). This gives the two sets of components that are to be assigned to buckets. Figure 9(b) gives the same two component sets using L, R terminology. Note that C_p corresponds to L_{j-p+1} and R_{p-i+1} as $w_p = wl_{j-p+1} = wr_{p-i+1}$. The minimum number of buckets needed is not affected by exchanging the two columns of Figure 9(a). So, the solution to Figure 9(a) is the same as that for Figure 9(c). Furthermore, inverting the two columns to get Figure 9(d) does not affect the minimum number of buckets. Figure 9(d) in L, R, notation is given in Figure 9(e). Hence, $T(i, j, k, w) = B(j-k, k-i+1, w)$.

Since B can be computed in $O((j-i+1)^2)$ time, $T(i, j, k, w)$, $i \le k \le j$ can be computed in $O((j-i+1)^2)$ time. Hence, we can compute $T(i, j, w)$ in $O((j-i+1)^2)$ time.

We are now ready to see how $S(i, j, h)$ may be computed. When only one folding is permitted, the minimum width of a height h layout is one of the values in the set $\{ w_i, \ldots, w_j \} \cup \{ w_u + w_v \mid i \le u < v \le j \}$. Let $d_1 < d_2 < \ldots < d_p$ be the distinct values in this set. Note that $p \le (j-i+1)^2$. Since $T(i, j, d_u) \ge T(i, j, d_{u+1})$, $1 \le u < p$, $S(i, j, h) = d_q$ where q is the least integer for which $T(i, j, d_q) \le h$. q can be found by performing a binary search in the range $[1, p]$. For each examined value s in this range, $T(i, j, d_s)$ is computed. If $T(i, j, d_s) > h$, then values $\le s$ are eliminated from the range. Otherwise, values $> s$ are eliminated. Since a binary search is used, at most

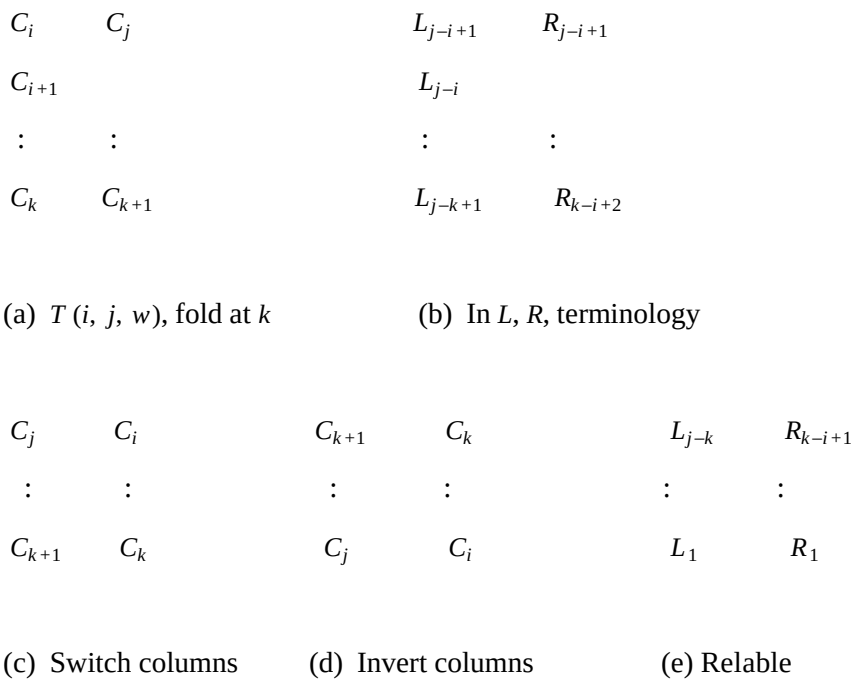


Figure 9: Equivalence.

$\lceil \log_2(p+1) \rceil$ s values are tried. Hence, at most this many $T(i, j, d_s)$'s are computed. So, $S(i, j, h)$ is computed in $O((j-i+1)^2 \log_2(j-i+1))$ time.

To compute $W(i, h)$ using (3') we need to compute at most $2h$ $S(i, j, h)$ values. Since $j - i < 2h$, this can be done in $O(h^3 \log h)$ time. To compute $W(1, h)$, we first compute $W(n, h)$, then $W(n-1, h)$, ... , and finally $W(1, h)$. The time for this is $O(h^3 n \log h)$. Since h need be at most n, the complexity is $O(n^4 \log n)$.

3.1.2 Width Of Folded Layout Is Fixed

When the width is fixed at w, we can find the minimum height layout in $O(n^4 \log^2 n)$ time by performing a binary search in the interval $[1, n]$. For each examined value h in this interval, we compute $W(1, h)$ as in the previous section. If $W(1, h) > w$, then heights $\leq h$ are eliminated. Otherwise, heights $> h$ are eliminated.

3.2 Routing Area At Stack Ends

Let r_i be as in Section 2.2. Define $T'(i, j, k, w)$ as below:

$$T'(i, j, k, w) = \begin{cases} T(i, j, k, w) + \max\{r_i, r_{j+1}\} + r_{k+1}, & k < j \\ T(i, j, k, w) + r_i + r_{j+1}, & k = j \end{cases}$$

$T'(i, j, k, w)$ is the height of a minimum height width w rectangle into which $C_i, C_{i+1}, \ldots, C_j$ can fit with fold at C_k. This height includes the needed routing space at the top and bottom of the component stacks. $T(i, j, k, w)$ is as defined in Section 3.1.1. If we use $T'(i, j, k, w)$ in place of $T(i, j, k, w)$ in the computation of $T(i, j, w)$ then the $S(i, j, w)$'s and $W(i, h)$'s computed in Section 3.1.1 account for the routing area. Hence, the minimum width height h folding that allows for routing area can be found in $O(h^3 n \log h)$ time. Similarly, the minimum height width w folding that accounts for routing area can be found in $O(n^4 \log^2 n)$ time.

4 Varible Width And Height Components

4.1 No Routing Area At Stack Ends

4.1.1 Height Of Folded Layout Is Fixed

Let h_i be the height of component C_i and let w_i be its width. Let $S(i, j, h)$ and $W(i,h)$ be as in Section 3.1.1. Using the same reasoning as in Section 3.1.1, we obtain

$$W(i, h) = \min_{i \leq j \leq n \,,\, \sum_{s=i}^{j} h_s \leq 2h} \{ S(i, j, h) + W(j+1, h) \} \qquad (5)$$

and $W(n+1, h) = 0$.

To obtain $S(i, j, h)$, we generalize P1 to the case where the L's and R's have possibly different heights. Let hl_s and hr_s , respectively, be the height of L_s and R_s. Let $B(i, j, w)$ be the minimum height rectangle into which the L_s's and R_s's can be placed so that:

(b') a horizontal line drawn at any vertical position of the rectangle cuts at most one member of L and at most one of R.

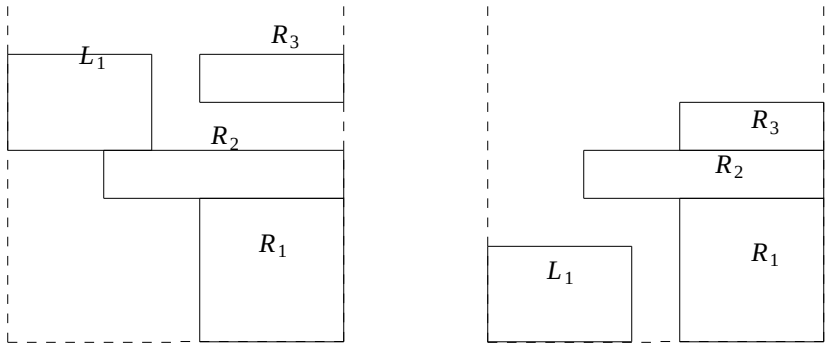
(c') if L_s and R_t are cut by some horizontal line, then $wl_s + wr_t \leq w$.

(d') the order in which the L_s is and R_s is appear in the rectangle bottom to top is $(L_1, \ldots, L_n)$ and $(R_1, \ldots, R_m)$, respectively.

If $wl_i + wr_j > w$ then using the reasoning of Section 3.1.1, we obtain:

$$B(i, j, w) = \min \{ hl_i + B(i-1, j, w) , hr_i + B(i, j-1, w) \}.$$

However, if $wl_i + wr_j \leq w$, the minimum $B(i, j, w)$ may not occur when L_i and R_j are placed adjacent to each other (see Figure 10).



(a) L_1 and R_3 adjacent

(b) L_1 and R_1 adjacent

Figure 10: L_1 and R_3 adjacent doesn't optimize B (1, 3, w).

When $wl_i + wr_j \leq w$, we need to compute i^* and j^* such that $wl_{i^*} + wr_{j^*} > w$. This is done as in Figure 11. To ensure proper termination of this code, we define $wl_0 = wr_0 = w + 1$ and $hl_0 = hr_0$

```
i* := i ;      j* := j ;
HL := hl_i ;   HR := hr_j ;
while wl_i* + wr_j* ≤ w do
    if  HL > HR
    then [ j* := j* − 1 ;   HR := HR + hr_j* ]
    else [ i* := i* − 1 ;   HL := HL + hl_i* ] ;
```

Figure 11: Computing i^* and j^*.

= 0. An example of this computation is given in Figure 12.

Let $B(0, j, w) = \sum_{s=1}^{j} hr_s$ and $B(i, 0, w) = \sum_{s=1}^{i} hl_s$. Theorem 2 establishes a recurrence for $B(i, j, w)$ when $wl_i + wr_j \leq w$.

Theorem 2: Let i and j be such that $wl_i + wr_j \leq w$ and let i^*, j^*, HL, and HR be as computed by the code of Figure 11. Then, $B(i, j, w) = \min \{ HL + B(i^*-1, j^*, w) , HR + B(i^*, j^*-1, w) \}$.

Proof: Since $wl_{i^*} + wr_{j^*} > w$, either L_{i^*} is above R_{j^*} or below it in every solution. If L_{i^*} is above R_{j^*} in an optimal solution (see Figure 13), then $HL_{opt} \geq HL = \sum_{s=i^*}^{i} hl_s$ as L_{i^*+1}, ..., L_i are above L_{i^*}. Furthermore, since at least L_{i^*-1}, ..., L_1 and R_{j^*}, ..., R_1 are below L_{i^*}, $H_{opt} \geq HL_{opt} + B(i^*-1, j^*, w) \geq HL + B(i^*-1, j^*, w)$. Since there is a feasible solution of height $HL + B(i^*-1, j^*, w)$ (by construction of Figure 11, R_j, ..., R_{j^*+1} can be packed in height HL adjacent to L_i, ..., L_{i^*}) and since H_{opt} is the minimum possible height of a feasible solution, it follows that $H_{opt} = HL + B(i^*-1, j^*, w)$. Similarly, if R_{j^*} is above L_{i^*} in the optimal solution, $H_{opt} = HR + B(i^*, j^*-1, w)$. Hence, $B(i, j, w) = H_{opt} = \min \{ HL + B(i^*-1, j^*, w) , HR + B(i^*, j^*-1, w) \}$. □

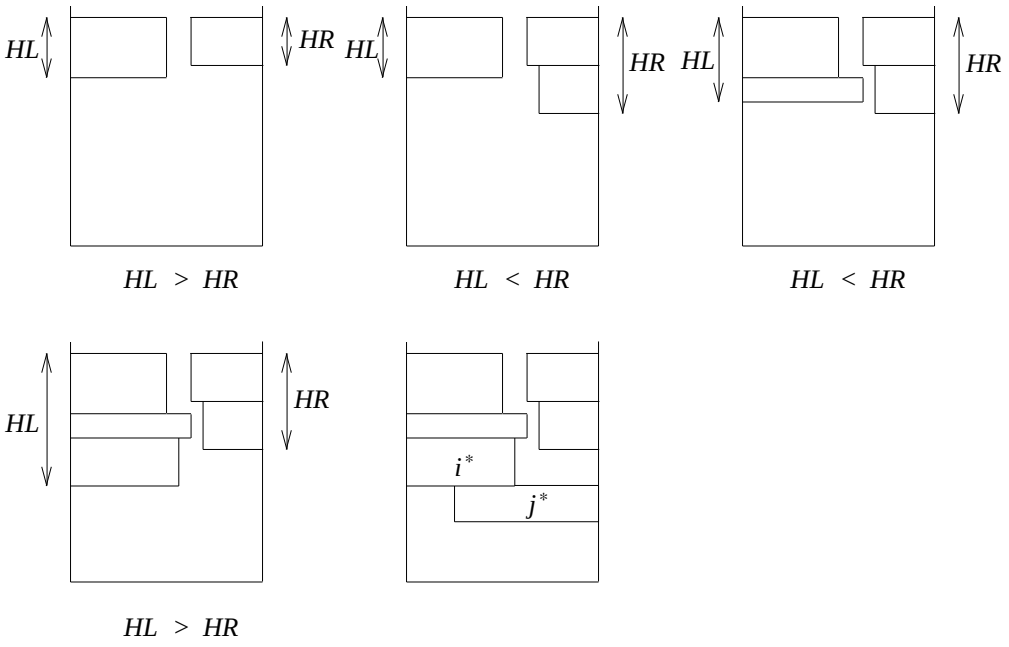


Figure 12: Example computation of i^* and j^*.

Since $|L| = n$ and $|R| = m$, a total of nm B values are to be computed. For each, we may need to compute i^* and j^* using Figure 12. This takes $O(n + m)$ time. So, while the B matrix could be computed in $O(nm)$ time when all components had the same height, it now takes $O(mn(n + m))$ time to do this. The remaining ideas of Section 3.1.1 directly carry over to the case of variable height and width components. The $T(i, j, k, w)$'s can be computed in $O((j - i + 1)^3)$ time for any fixed i and j and all k, $i \leq k \leq j$. Hence, each $T(i, j, w)$ can be obtained in $O((j - i + 1)^3)$ time. So, each $S(i, j, h)$ can be obtained in $O(h^3 \log h)$ time. As a result, (5) can be solved for $W(1, h)$ in $O(h^4 n \log h)$ time.

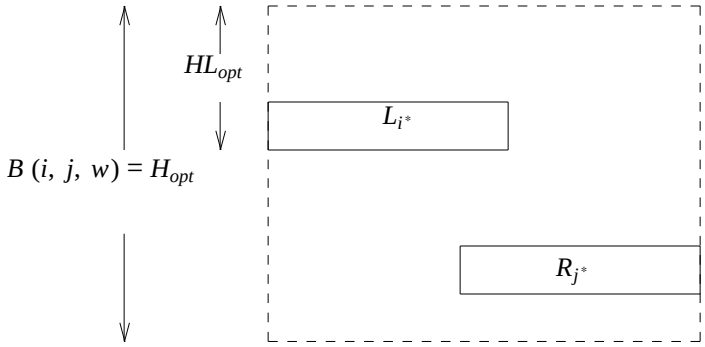


Figure 13: L_{i^*} above R_{j^*} in optimal solution.

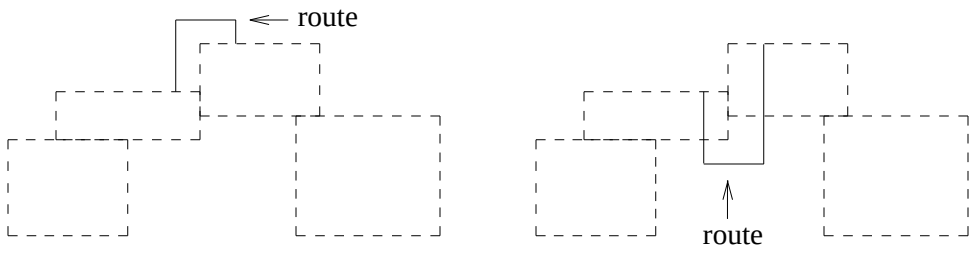
4.1.2 Width Of Folded Layout Is Fixed

The binary search technique of Section 3.1.2 may be used to obtain the optimal height solution in $O(n^5 \log^2 n)$ time as the optimal height is one of the n^2 values $h(i, j)$, $1 \leq i \leq j \leq n$.

4.2 Routing Area At Stack Ends

We assume that all inter stack routes are done at the top or bottom ends of the stacks as in Figure 14(a), rather than in space internal to the stack as in Figure 14(b). More specifically, if R is the minimum height rectangle into which $C_i, C_{i+1}, \ldots, C_j$ can be folded using at most one folding position then all inter stack routing is done external to R.

The technique of Section 4.2 readily generalizes to the case of varible height and width components. The complexity of the algorithms for this case are the same as those for the case when no routing area is needed at the stack ends.



(a) legal route (b) illegal route

Figure 14:

5 Folding Without Nesting

Our folding model permits the slices of one stack to nest with the slices of one adjacent stack. For example, in Figure 1(b) slice 4 of stack 1 occupies the same physical space as slice 4 of stack 2. As noted earlier, this nesting of stacks may increase the demand for inter component routing tracks. In situations where increased routing tracks cannot be provided, one may forbid stack nesting. So, the stacks of Figure 1(b) will have to be placed as in Figure 15. In this section we consider folding without nesting. Since nesting can occur only when the component widths are not the same, we need only consider this case.

5.1 Height Constrained

Let $W(i)$ be the minimum width rectangle of height h into which the components $C_i, \ldots, C_n$ can be folded without nesting. The correctness of the following recurrence is easily established.

$$W(i) = \min_{i \le j \le n \ , \ h(i,\, j) \le h} \{ \max_{i \le q \le j} \{w_q\} + W(q+1) \}, \qquad 1 \le i \le n$$

$$W(n+1) = 0$$

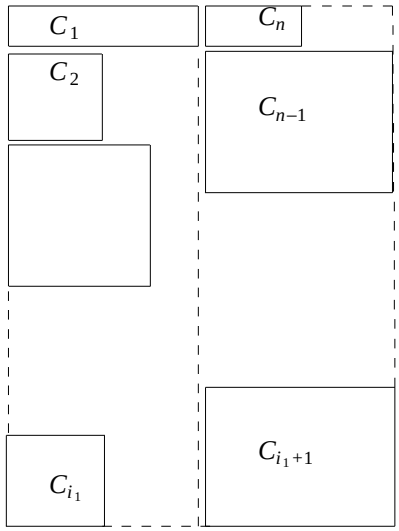


Figure 15: Folding without nesting.

This recurrence may be solved for $W(1)$ in $O(n^2)$ time.

5.2 Width Constrained

Since the optimal height is one of the n^2 values $h(i, j)$, $1 \leq i \leq j \leq n$, we can perform a binary search over these heights to determine the smallest height that results in folding of width no more than permissible. For each tested height the $O(n^2)$ height constrained algorithm is used. The overall complexity is $O(n^2 \log n)$.

6 Conclusions

We have considered the problem of folding bit sliced stacks so as to obtain minimum height (subject to width constraints) and minimum width (subject to height constraints) foldings. Our model differs from that of [LARM90] in that we do not permit a reordering of the components in the

input component stack. Our model applies to bit sliced architectures as well as to standard cell and sea-of-gates designs.

Polynomial time algorithms to obtain optimal foldings have been obtained under a variety of assumtions (stack nesting permitted/not permitted, routing space needed/not needed at stack ends, equal height components, equal width components). Our algorithms for the stack nesting case are summarized below.

Stack nesting permitted	Routing area at stack ends	
	No	yes
Equal width, height constrained	$O(n)$	$O(n^2)$
Equal width, width constrained	$O(n^2)$	$O(n^3)$
Equal height, height constrained	$O(n^4 \log n)$	$O(n^4 \log n)$
Equal height, width constrained	$O(n^4 \log^2 n)$	$O(n^4 \log^2 n)$
Variable heights and widths, height constrained	$O(n^5 \log n)$	$O(n^5 \log n)$
Variable heights and widths, width constrained	$O(n^5 \log^2 n)$	$O(n^5 \log^2 n)$

When stack nesting is not permitted, the height constrained case may be solved in $O(n^2)$ time and the width constrained case in $O(n^2 \log n)$ time.

7 References

[HORO78] E. Horowitz, and S. Sahni, "Fundamentals of Computer Algorithms", Computer Science Press, Maryland, 1978.

[LARM90] L. Larmore, D. Gajski and A. Wu, "Layout Placement for Sliced Architecture," University of California, Irvine, Technical Report, 1990.

[Wu90] A. Wu, and D. Gajski, "Partitioning Algorithms for Layout Synthesis from Register-Transfer Netlists," *Proc. of International Conference on Computer Aided Design*, November 1990, pp. 144-147.

[SHRA88] E. Shragowitz, L. Lin, S. Sahni, "Models and algorithms for structured layout," *Computer Aided Design*, Butterworth & Co, 20, 5, 1988, 263-271

[SHRA90] E. Shragowitz, J. Lee, and S. Sahni, "Placer-router for sea-of-gates design style," in *Progress in computer aided VLSI design*, Ed. G. Zobrist, Ablex Publishing, Vol 2, 1990, 43-92