INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Denser Packings Obtained in O(n log log n) Time

David Pisinger,

To cite this article:
David Pisinger, (2007) Denser Packings Obtained in O(n log log n) Time. INFORMS Journal on Computing 19(3):395-405.
https://doi.org/10.1287/ijoc.1060.0192

INFORMS Journal on Computing
Vol. 19, No. 3, Summer 2007, pp. 395–405
ISSN 1091-9856 | EISSN 1526-5528 | 07 | 1903 | 0395

DOI 10.1287/ijoc.1060.0192

Denser Packings Obtained in $O(n \log \log n)$ Time

David Pisinger
DIKU, University of Copenhagen, DK-2100 Copenhagen, Denmark,
pisinger@diku.dk

The placement problem is that of packing a set of rectangles into a minimum-area enclosing rectangle. Since it is difficult to optimize directly on a placement, a number of topological representations have been presented in the literature. One of the most successful is the sequence pair representation. As opposed to previous papers using sequence pair, we do not make use of the corresponding constraint graph, but interpret the sequences as a packing order in a constructive algorithm. All placements generated are semi-normalized, i.e., each module is moved as far left and down as possible according to the current packing contour. It is shown that the two interpretations are equivalent for any minimum-area placement, and for a given sequence pair the new interpretation results in a placement using no more area than the constraint graph interpretation. The transformation runs in $O(n \log \log n)$ time and it is able to handle various constraints on the location of modules. Computational results based on a simulated-annealing framework show that the algorithm is able to improve significantly on the best found solutions for benchmark very large-scale integrated (VLSI) circuit design problems using less than ten seconds of CPU time. For large packing problems it is able to find solutions quickly that waste no more than 3%–5% of the space.

Key words: programming, integer, algorithms, heuristic; analysis of algorithms: computational complexity; production-scheduling, cutting stock
History: Accepted by William J. Cook, Area Editor for Design and Analysis of Algorithms; received July 2004; revised September 2005, March 2006; accepted May 2006. Published online in *Articles in Advance* July 20, 2007.

1. Introduction

One of the most critical stages in very large-scale integrated (VLSI) layout is the placement problem in which a given set of rectangular modules have to be placed within a rectangular domain so as to minimize the area used. Minimizing the chip area is crucial to obtain good yield—not only will a small area result in an increased production of chips per wafer, but also the relative number of erroneous chips will decrease. The placement problem can be formulated as a *two-dimensional minimum-area packing problem*:

Given n rectangular modules $m_1, \ldots, m_n$, module j having width w_j and height h_j, the objective is to assign coordinates $x_j, y_j \geq 0$ to each module j such that no two modules overlap and such that the enclosing rectangle is minimized. The coordinates (x_j, y_j) of a module j refer to the lower left corner of the module.

The problem is $\mathcal{NP}$-hard to solve (Christofides and Whitlock 1977), and hence heuristics must be applied for solving large instances. Most of the heuristics presented during the last decade use some kind of abstract representation of a placement in combination with a local search algorithm. Among the most successful representations are sequence pair (Murata et al. 1996), O-tree (Guo et al. 1999, Pang et al. 2000), B*-tree (Chang et al. 2000) and corner block list (Hong et al. 2000).

The sequence pair representation has the advantage that it is a real topological representation, and that efficient algorithms exist to transform the abstract representation to a concrete placement. A placement of n modules is represented by a pair of sequences given as permutations of the numbers $\{1, \ldots, n\}$. For every sequence pair the corresponding block placement can be found in $O(n^2)$ time by constructing a horizontal and vertical constraint graph and computing longest paths in both of the graphs. Takahashi (1996) improved the time complexity of this task to $O(n \log n)$, although it was only shown how to derive the size of the enclosing rectangle and not the individual coordinates of the modules. Tang et al. (2000) presented an algorithm solving both tasks in $O(n \log n)$ time based on the determination of longest common subsequences in the horizontal and vertical constraint graphs. In a recent paper Tang and Wong (2001) showed how the time complexity of the same algorithm may be decreased to $O(n \log \log n)$ by using more advanced data structures (van Emde Boas 1977). The improved algorithm was also extended to handle pre-placed constraints, range constraints, and boundary constraints.

A disadvantage of all the above algorithms based on sequence pairs is that the obtained placements are not *normalized*, i.e., modules are not moved as far left

and down as possible. It is obvious that a minimum-area placement will be normalized (or an equivalent normalized placement exists), and hence it should be more advantageous to use local search algorithms on normalized placements. This paper introduces a new interpretation of sequence pairs that leads to semi-normalized placements. Although the new interpretation does not fully respect the horizontal and vertical constraint graph corresponding to a sequence pair, it is shown that for a minimum-area placement the two interpretations are equivalent. The developed algorithm for transforming a sequence pair to a placement runs in $O(n \log \log n)$ time, matching the time complexity of the FAST-SP algorithm (Tang and Wong 2001). The algorithm is very simple and hence the constants hidden in the big-Oh notation are indeed small. Computational results are presented showing that several million placements can be investigated in less than ten seconds for the largest MCNC benchmark problems.

The algorithm presented in this paper has roots in packing techniques for two- and three-dimensional bin-packing problems (Martello et al. 2000, Martello and Vigo 1998) where a contour-building approach frequently is used. The considered placement problem is closely related to the *strip-packing problem* in which one of the sides of the enclosing rectangle has fixed dimension, and where the objective is to minimize the other side. See Martello et al. (2003) and Iori et al. (2003) for recent exact and heuristic algorithms for the strip-packing problem.

Packing rectangles into the smallest enclosing rectangle is also related to several scheduling problems as noted by Korf (2004). Assume that we have a set of independent and indivisible jobs, each requiring a certain number of workers for a certain time. The smallest enclosing rectangle minimizes the overall labor costs as a product of workers and labor time.

In Section 2 we give a formal definition of the placement problem and the sequence pair representation using properties from bin-packing problems. In Section 3 we present a new interpretation of the sequence pair representation and show how a semi-normalized placement can be obtained in $O(n \log \log n)$ time. Section 4 compares the two transformations showing that the latter leads to solutions of higher quality, and Section 5 describes the corresponding simulated annealing algorithm. Handling of various constraints on the placement of modules is discussed in Section 6. Finally, Section 7 brings the computational results for MCNC benchmarks and for various instances from the packing world.

2. Sequence Pair

A placement is *normalized* if all modules are adjusted as far left and down as possible without overlapping. Christofides and Whitlock (1977) showed that any two-dimensional placement can be replaced by an equivalent placement where no item may be moved leftwards or downwards without extending the area used.

Assume that we place the modules in some sequence $\langle i_1, i_2, \ldots, i_n \rangle$. Having placed the first k modules $i_1, i_2, \ldots, i_k$ we may define the *contour* E_k of the placement as the area below or left of the placed modules. More formally, $E_k = \{(x', y') \geq 0: \exists i \in \{i_1, i_2, \ldots, i_k\}$ such that $x' \leq x_i + w_i$ and $y' \leq y_i + h_i\}$. If two modules i and j have been placed, we say that module i *shades* module j if $x_j + w_j \leq x_i + w_i$ and $y_j + h_j \leq y_i + h_i$. Having placed the first k modules, the *extreme modules* are those that define the contour, i.e., a module i is extreme if it is not shaded by any $j \in \{i_1, \ldots, i_k\}$.

A placement is *semi-normalized* if there exists an ordering $\langle i_1, i_2, \ldots, i_n \rangle$ of the modules such that module i_{k+1} is placed as far left and down as possible without intersecting the contour E_k. Figure 1 illustrates the definitions: (a) is a placement of $n = 7$ modules, while (b) is the corresponding normalized placement. In (c) the contour E_e is marked with a dashed line after rectangles a, b, c, d, e have been placed. Modules d, c, e are extreme. Module c shades module b but it does not shade module a. On the other hand module e shades module a. Finally, (d) shows a semi-normalized placement corresponding to the order a, b, c, d, e, f, g. Notice that module f is placed as far left and down as possible with respect to the contour E_e.

Murata et al. (1996) presented a very elegant representation of block placements based on sequence pairs.

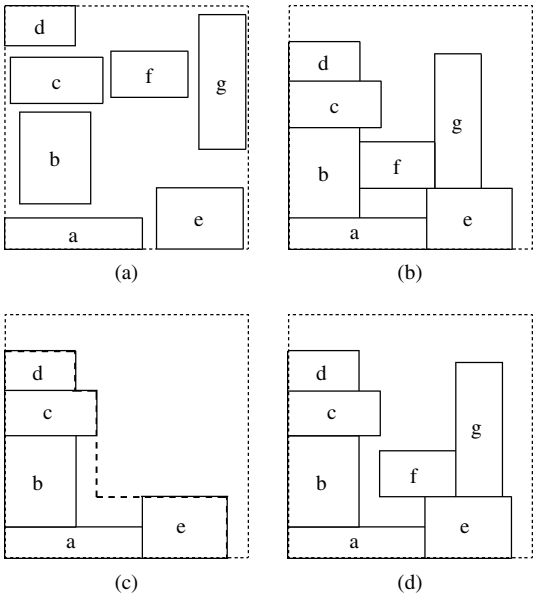


Figure 1 **(a–d) Illustration of Normalized and Semi-Normalized Placements**

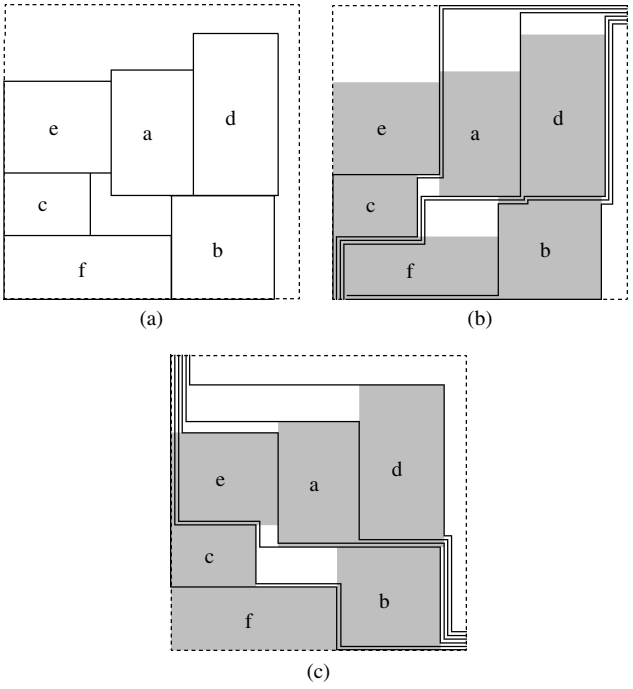


Figure 2 **(a–c) A Placement and the Corresponding Positive and Negative Step-Lines**

Every placement can be represented by two permutations of the numbers $\{1, 2, \ldots, n\}$, a so-called sequence pair $(A, B) = (\langle a_1, a_2, \ldots, a_n \rangle, \langle b_1, b_2, \ldots, b_n \rangle)$. Sequence A is called the *positive sequence* while B is called the *negative sequence*.

For a given placement, the two permutations A and B are found as follows: First we draw a sequence of *positive steplines* that separates the modules from each other. A positive stepline is a line starting from the lower left corner of the placement, and finishes at the upper right corner. The steplines $L_1, \ldots, L_n$ must not intersect, and each stepline separates one more module from the rest. Figure 2(a) shows a placement and (b) shows the sequence of positive steplines. L_1 separates module e, L_2 separates module c, etc. (for clarity some of the steplines have been slightly moved to avoid overlap). This results in the sequence $A = \langle \mathtt{e}, \mathtt{c}, \mathtt{a}, \mathtt{d}, \mathtt{f}, \mathtt{b} \rangle$. A negative stepline starts in the upper left corner, and ends in the lower right corner. The sequence of steplines should separate one module in each step as illustrated in Figure 2(c), which results in the sequence $B = \langle \mathtt{f}, \mathtt{c}, \mathtt{b}, \mathtt{e}, \mathtt{a}, \mathtt{d} \rangle$.

We observe that if module i precedes j in A, then

$$x_i + w_i \le x_j \quad \text{or} \quad y_j + h_j \le y_i \tag{1}$$

i.e., module i is left of or above module j. Similarly, if i precedes j in B, then module i is left of or below module j, hence

$$x_i + w_i \le x_j \quad \text{or} \quad y_i + h_i \le y_j. \tag{2}$$

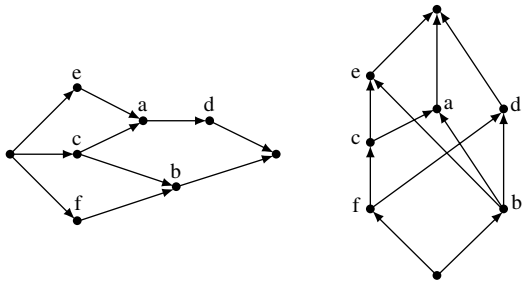


Figure 3 **Constraint Graphs Corresponding to the Sequence Pair** $(A, B) = (\langle \mathtt{e}, \mathtt{c}, \mathtt{a}, \mathtt{d}, \mathtt{f}, \mathtt{b} \rangle, \langle \mathtt{f}, \mathtt{c}, \mathtt{b}, \mathtt{e}, \mathtt{a}, \mathtt{d} \rangle)$

From the definitions we immediately get

$$(\langle ..m_i..m_j.. \rangle, \langle ..m_i..m_j.. \rangle) \Rightarrow m_i \text{ is left of } m_j \tag{3}$$

$$(\langle ..m_j..m_i.. \rangle, \langle ..m_i..m_j.. \rangle) \Rightarrow m_i \text{ is below } m_j. \tag{4}$$

Hence, if module i precedes module j in both sequences, then i must be placed left of j. In a similar way if i succeeds j in sequence A but i precedes j in sequence B then i must be placed below j.

The relations (3) and (4) can be used to derive a constraint graph as illustrated in Figure 3 (redundant edges have been removed for clarity). Edges indicate which modules should be placed left of each other (respectively below each other). Traversing the nodes in topological order while assigning coordinates to the modules, a placement can be obtained in $O(n^2)$ time. We will denote this algorithm by SP. Tang et al. (2000) and Tang and Wong (2001) showed how the same placement can be derived without explicitly defining the constraint graph, but by finding weighted longest common subsequences in the sequence pair. The latter algorithm will be denoted FAST-SP.

3. New Transformation

We will now present a new transformation from sequence pairs to placements that has the property that the returned placements are semi-normalized, and fit in a smaller area. The algorithm is based on packing principles from Martello et al. (2000) where we use the sequence pair (A, B) to define the packing order and to define the packing points.

Assume that the sequence pair is given as $(A, B) = (\langle a_1, a_2, \ldots, a_n \rangle, \langle b_1, b_2, \ldots, b_n \rangle)$, where both A and B are permutations of $\{1, \ldots, n\}$. For simplicity we add two artificial modules $\mathtt{s}$ and $\mathtt{t}$ having dimensions $(w, h) = (0, \infty)$ and $(\infty, 0)$, respectively. The two modules are placed at positions $(x_\mathtt{s}, y_\mathtt{s}) = (x_\mathtt{t}, y_\mathtt{t}) = (0, 0)$. We also add the two modules to sequence A so that it becomes $A = \langle \mathtt{s}, a_1, a_2, \ldots, a_n, \mathtt{t} \rangle$. At any iteration of the algorithm we let E denote the set of extreme modules. The set of extreme modules E is implemented as a double-linked list (see Figure 4), and modules in the list have the same order as in sequence A.

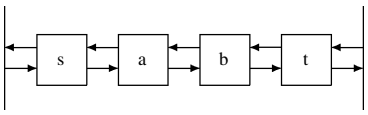


Figure 4 **Double-Linked List Used to Implement the Set of Extreme Modules**

1 **algorithm** SEMINORM-SP($A = \langle a_1, \ldots, a_n \rangle$, $B = \langle b_1, \ldots, b_n \rangle$)
2 $E = \{\mathtt{s}, \mathtt{t}\}$
3 **for** $k = 1$ **to** n **do**
4 $h = b_k$
5 $i = \mathtt{predecessor}(E \cap A, h)$
6 $j = \mathtt{successor}(E \cap A, h)$
7 $x_h = x_i + w_i$; $y_h = y_j + h_j$ //place module h in the corner formed by modules i and j
8 $E = E \cup \{h\}$ //insert module h after module i in double-linked list
9 remove rectangles shadowed by module h from E

Figure 5 shows the six iterations of algorithm SEMINORM-SP for the sequence pair $(A, B) = (\langle \mathtt{e}, \mathtt{c}, \mathtt{a}, \mathtt{d}, \mathtt{f}, \mathtt{b} \rangle, \langle \mathtt{f}, \mathtt{c}, \mathtt{b}, \mathtt{e}, \mathtt{a}, \mathtt{d} \rangle)$. In each iteration the set $E \cap A$ is represented by the underlined letters in the list $A = \langle \mathtt{s}, \mathtt{e}, \mathtt{c}, \mathtt{a}, \mathtt{d}, \mathtt{f}, \mathtt{b}, \mathtt{t} \rangle$. The current contour is marked with a dashed line.

The double-linked representation of E makes it easy to remove shadowed modules in step 9 of the algorithm. As we know the two modules i and j between which module h is placed, we simply follow the link from j, forward testing whether the module is shaded by module h. As soon as a module has been found that is not shaded by module h we do not need to follow the link further. Similarly, we follow the link from i, backward testing whether some modules are shaded. In iteration 6 of Figure 5, having inserted module d between modules a and b we follow the link from b forward (removing module b and keeping t), and we follow the link backward from a (removing module a and keeping s).

Steps 5 and 6 of the algorithm are implemented using a search tree supporting the operations insert, delete, predecessor. The search tree holds the current modules in $E \cap A$, using the position i of a module a_i in sequence $A = \langle a_1, a_2, \ldots, a_n \rangle$ as search key. The operation predecessor($E \cap A, h$) returns the module in sequence $E \cap A$ preceding module h. In iteration 5 of Figure 5 the set E will look as illustrated in Figure 4. The sequence $E \cap A$ is given by the underlined letters in $\langle \underline{\mathtt{s}}, \mathtt{e}, \mathtt{c}, \underline{\mathtt{a}}, \mathtt{d}, \mathtt{f}, \underline{\mathtt{b}}, \underline{\mathtt{t}} \rangle$. Calling predecessor($E \cap A$, d) returns a as this is the underlined letter preceding d. Notice that we do not explicitly need the successor operation in line 6 of the algorithm: Having the predecessor j we may find the successor in constant time using the linked list of extreme modules (Figure 4).

Each call to insert, delete or predecessor can be made in $O(\log \log n)$ time using van Emde Boa trees (van Emde Boas 1977). To place a rectangle takes $O(1)$ time, while the removal of shadowed rectangles may be bounded using amortized arguments. Placing n rectangles can result in the removal of at most $n - 1$ shadowed rectangles, hence the amortized time complexity for this part is $O(1)$ per placed rectangle. The total time for placing n rectangles is hence $O(n \log \log n)$.

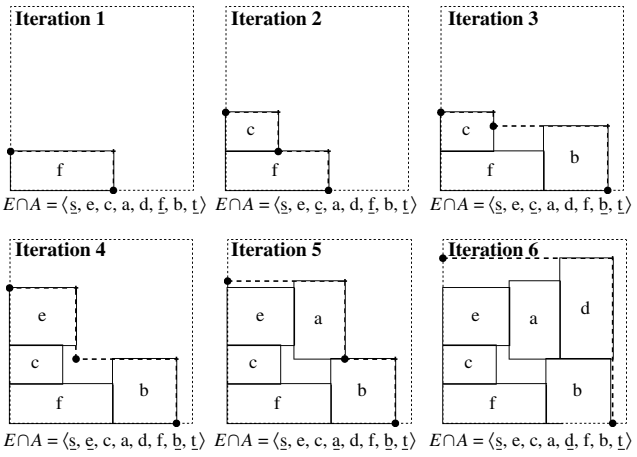


Figure 5 **Illustration of the SEMINORM-SP Algorithm**

4. A Comparison of the Two Transformations

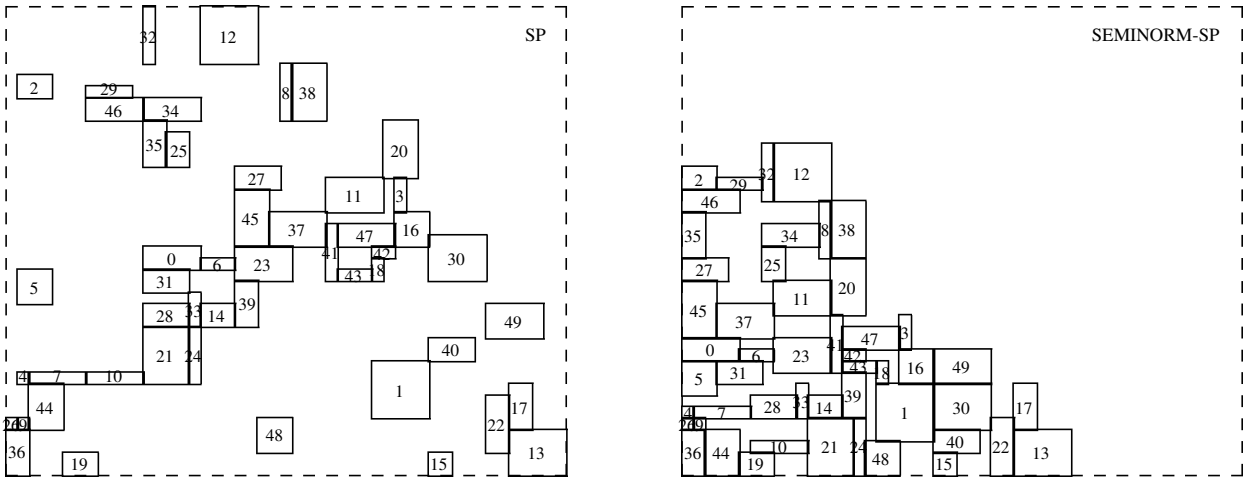
For a randomly generated sequence pair with $n = 50$ modules, Figure 6 shows the difference between the SP and the SEMINORM-SP transformations. The figure clearly illustrates that the semi-normalized placement is much denser for the given instance. Indeed we have the following general result:

PROPOSITION 1. *For any sequence pair (A, B) the placement obtained by* SEMINORM-SP *never takes up more space than the placement obtained by* SP.

PROOF. For a given sequence pair (A, B), assume that the modules were placed by SP, draw the steplines corresponding to (A, B) on the found placement, and denote these steplines the SP-steplines. Similarly, place the modules by SEMINORM-SP, and draw the steplines corresponding to (A, B) on the found placement. Denote these steplines the SEMINORM-SP-steplines.

We say that a negative stepline L_1 *dominates* another negative stepline L_2 if it never is above the latter line.

We wish to prove by induction that the negative SEMINORM-SP-steplines dominate the corresponding negative SP-steplines. It is obvious that after placing the first module, the two steplines are the same. Now assume that the negative SEMINORM-SP-steplines dominate the corresponding negative SP-steplines for modules up to module $k - 1$.



((26, 2, 5, 4, 7, 29, 46, 32, 34, 10, 35, 25, 0, 12, 6, 27, 45, 8, 38, 37, 31, 23, 28, 11, 20, 41, 47, 43, 3, 16, 42, 18, 33, 14, 30, 39, 21, 24, 9, 49, 44, 40, 36, 1, 48, 22, 19, 15, 17, 13),
(36, 26, 9, 44, 19, 4, 7, 10, 21, 28, 5, 24, 33, 14, 31, 0, 39, 6, 48, 23, 45, 37, 27, 35, 46, 41, 43, 1, 15, 2, 29, 18, 42, 47, 16, 11, 40, 30, 25, 22, 3, 20, 13, 34, 8, 17, 49, 38, 32, 12))

Figure 6 Transformation of a Sequence Pair to a Placement Using SP and SEMINORM-SP

When we place module k the negative SEMINORM-SP-stepline may look like the black line in Figure 7(a), and we may assume that module k should have been placed at the marked position if we used algorithm SP. We do not know how the SP-stepline looks, but it must at least cover module k and due to the induction assumption, it must be equal to or above the SEMINORM-SP-stepline. This "minimum" SP-stepline is shown as the gray line in (a). If we can prove that SEMINORM-SP will place module k within the minimum SP-stepline, we are done.

Hence, assume that k is placed above the minimum SP-stepline as illustrated in Figure 7(b). Let j be the module directly below k. Consider Step 5 and 6 of the SEMINORM-SP algorithm. Module j is an extreme module, hence it can be found in list E, and j must succeed k in sequence A. On the other hand, we know that j precedes k in sequence B since j was placed before module k. By using criteria (3) and (4) we note that j must be below k in the SP placement. But this means that SEMINORM-SP placed j above its SP-position, contradicting our induction assumption.

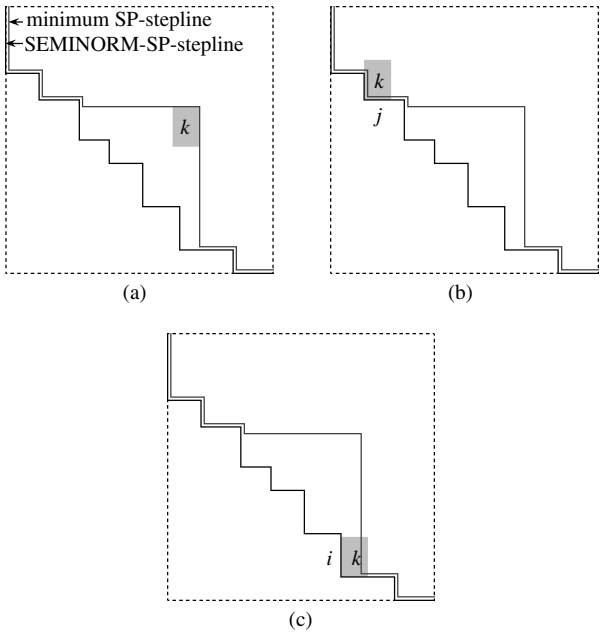


Figure 7 (a–c) Negative Steplines for SP and SEMINORM-SP with Possible Placements of k

Similarly, we may assume that k is placed right of the minimum SP-stepline as illustrated in Figure 7(c). As before, let i be the module directly left of k. Using the same arguments as above, we know from criteria (3) and (4) that i must be left of k in the SP placement. But this means that SEMINORM-SP placed i right of its SP-position, contradicting our induction assumption.

Since the negative SEMINORM-SP-steplines dominate the corresponding negative SP-steplines in every step, this also holds for the last steplines, and hence the area used by SEMINORM-SP is not larger than that of SP. □

PROPOSITION 2. *Assume that $\mathcal{P}$ is a normalized placement represented by the sequence pair (A, B). Then both* SEMINORM-SP *and* SP *with (A, B) as input will return the placement $\mathcal{P}$.*

PROOF. It is well known that SP returns the correct placement $\mathcal{P}$, hence we need to show that SEMINORM-SP also returns placement $\mathcal{P}$. The proof is done by induction in the number of modules placed. Obviously SEMINORM-SP returns the correct placement for the first module. Now assume that SEMINORM-SP has placed all modules $1, \ldots, k-1$ at the correct positions, and we wish to prove that module k is placed at the correct position. From the proof of the previous proposition, we know that SEMINORM-SP will place module k within the SP-stepline. On the other

hand, since $\mathcal{P}$ is normalized, module k is supported from left and below other modules (in SP, and also in SEMINORM-SP due to the induction assumption). Hence module k can only be placed at the same position as found by SP. □

Since our objective is to find a minimum-area placement (i.e., a normalized placement), Proposition 2 states that we may use algorithm SEMINORM-SP instead of SP for evaluating the placements, obtaining the same optimal solution. Using SEMINORM-SP instead of SP in a heuristic (and assuming that the two heuristics traverse the same subset of sequence pairs) will frequently result in a smaller solution value found, since SEMINORM-SP returns a denser placement in every step of the heuristic as shown in Proposition 1.

Algorithm SEMINORM-SP differs from SP by the fact that we have relaxed the relations (1) imposed by sequence A and use sequence A only for relative positioning of the modules. Sequence B is, however, strictly respected with respect to (2). Based on this observation, one could derive an equivalent algorithm respecting sequence A and relaxing sequence B. Indeed one could hope that the two algorithms could complement each other well, and that alternations between the two versions would lead to even tighter placements.

5. Simulated Annealing

A simulated annealing algorithm using sequence pair as representation was implemented, where SEMINORM-SP was used to evaluate the area consumption $f(x)$ of a given solution x. In each step of the algorithm one of three moves takes place to obtain a new solution x' from the present solution x: Either a randomly chosen module is rotated, or two modules are interchanged in sequence A or two modules are interchanged in sequence B. The new sequence pair is evaluated using algorithm SEMINORM-SP. If a better solution is obtained, i.e., $f(x') < f(x)$, the new solution is accepted. Otherwise we evaluate

$$P_{acc}(x \to x') = e^{-(f(x')-f(x))/T}. \tag{5}$$

A random number p, between 0 and 1 is drawn, and if $p < P_{acc}$, the new solution x' is accepted as the present solution. The parameter T in (5), known as the *temperature*, is gradually decreased according to an exponential function.

To speed up the algorithm the traditional simulated annealing framework was modified as follows. The random number p was derived before transforming the sequence pair to a placement. From (5) an upper bound u was derived, giving the maximum value of $f(x')$ in order to accept x' as the current solution. Knowing u, the SEMINORM-SP algorithm could be aborted whenever the placement of some rectangles made it clear that $f(x')$ would exceed u, i.e., if the area of the enclosing rectangle at a given stage exceeds the threshold value. The approach made it possible to terminate the transformation before completion in around 90% of the evaluations.

The start-temperature was chosen as $T_0 = 0.1A_T$, where $A_T = \sum_{j=1}^{n} w_j h_j$ is the total sum of the module areas. The number of iterations in the simulated annealing was set to depend linearly on n as $100{,}000n$. The temperature was decreased using the exponential cooling scheme $T := 0.9995T$ every $5n$ iterations.

6. Handling Additional Constraints

In VLSI design a number of additional constraints may be imposed on the placement of specific modules (Tang and Wong 2001). This includes *pre-placed modules*, which should be placed with their lower left corner (x_j, y_j) at position (x'_j, y'_j), and *modules with range constraints*, which should be placed with their lower left corner (x_j, y_j) in an interval satisfying $x_j^1 \le x_j \le x_j^2$ and $y_j^1 \le y_j \le y_j^2$. Notice that pre-placed modules are a special case of range-constrained modules, and hence only the latter will be discussed in the following. Also notice that modules without range constraints can be assigned the range $(x_j^1, x_j^2) = (y_j^1, y_j^2) = (0, \infty)$, so we may assume that all modules have a range constraint.

To handle range-constrained modules we replace line 7 in algorithm SEMINORM-SP with the following

7.a $x_h = x_i + w_i$; $y_h = y_j + h_j$ //place module h in the corner formed by modules i and j
7.b **if** $x_h > x_h^2$ or $y_h > y_h^2$ **return false**
7.c $x_h = \max(x_h, x_h^1)$; $y_h = \max(y_h, y_h^1)$

Line 7.a corresponds to the old line 7. Line 7.b checks whether the current sequence pair is able to respect the range constraints on module h. If SEMINORM-SP returns **false** in this line the local search framework should reject or punish the present solution. Line 7.c lifts module h upward right such that it satisfies the range constraints. Note that in line 9 of SEMINORM-SP we should use the lifted coordinates of module h when removing shadowed modules.

As can be seen, the additional tests do not affect the asymptotic running time, and the algorithm returns **false** only if no placement satisfying the range constraints exists for the given sequence pair.

Finally, we are able to ensure a given *aspect ratio* ρ between the width W and height H of the enclosing rectangle. At any stage of the SEMINORM-SP algorithm we may increase the dimensions of the enclosing rectangle as $W := \max(W, H/\rho)$ and $H := \max(H, W/\rho)$, which ensures that $\max(W/H, H/W) \le \rho$. If the resulting area WH exceeds the threshold value u

Table 1 Characteristics of the MCNC General Cell Circuits

	Modules		Pins			
Circuit	Free	Fixed	Non-I/O	I/O	Nets	Module area (mm^2)
apte	9	0	287	73	97	46.562
xerox	10	0	698	2	183	19.350
hp	11	0	309	45	71	8.831
ami33	33	0	520	40	122	1.156
ami49	49	0	953	22	396	35.445

described in the end of Section 5, we may immediately reject the current sequence pair as it will not be accepted by the simulated annealing algorithm.

7. Computational Results

The presented algorithm was implemented in C and run on a PC with an AMD64 2.4 GHz processor using the gcc compiler version 3.3.3. The computational results were carried out on the MCNC general cell benchmark problems (MCNC 2001), the bin-packing instances proposed by Berkey and Wang (1987) and Martello and Vigo (1998), and finally the square packing instances proposed by Korf (2004).

7.1. MCNC Instances

There are five circuits in the MCNC benchmark set as described in Table 1. Rotations are allowed for the modules and the placement area contains no blockages or fixed modules.

Table 2 compares several of the most promising general cell placement algorithms with respect to running time and solution quality. The results of O-tree, B*-tree, and FAST-SP are taken directly from Tang and Wong (2001). The running times of SEMINORM-SP are total running times of the whole simulated annealing algorithm for the best run out of 20. The variation in the found solution is quite small, though.

SEMINORM-SP in no case is using more than ten seconds of CPU time and it finds the best solutions reported in the literature. For large instances it improves solution quality significantly. For the ami33 instance SEMINORM-SP reduced the wasted space from best known 4.2% to quite impressive 1.1%. For the ami49 instance the wasted space was reduced from 3.0% to 2.0%.

Table 2 Results on MCNC Instances

	O-tree	B*-tree	FAST-SP		SEMINORM-SP		
Circuit	Area (mm^2)	Area (mm^2)	Area (mm^2)	Waste (%)	Area (mm^2)	Time (s)	Waste (%)
apte	46.92	46.92	46.92	0.77	46.92	0.58	0.77
xerox	20.21	19.83	19.80	2.32	19.80	0.68	2.29
hp	9.159	8.95	8.947	1.30	8.947	0.75	1.30
ami33	1.242	1.27	1.205	4.20	1.169	4.61	1.11
ami49	37.73	36.80	36.50	2.98	36.18	9.85	2.04

Note. Solutions for SEMINORM-SP are best out of 20 runs.

Table 3 Results on MCNC Instances Using 300 Times More Iterations in the Simulated Annealing

	SEMINORM-SP		
Circuit	Area (mm^2)	Time (s)	Waste (%)
ami33	1.16796	1,359	0.99
ami49	35.9936	3,004	1.52

7.2. MCNC Instances with Longer Running Times

To investigate the quality of SEMINORM-SP when given a larger running time, we increased the number of iterations in the simulated annealing algorithm by a factor of 300. The resulting solutions are in Table 3. The wasted space of ami33 and ami49 was further reduced to 0.99% and 1.5%, respectively. The two found solutions are depicted in Figures 8 and 9.

It should also be noted that the MCNC general cell instances are very small, and do not fully show the benefits of SEMINORM-SP. For very large instances, where only a limited number of iterations can be performed in the local search, an algorithm cannot rely on finding a normalized placement by local exchanges—normalization must be an inherent property of the algorithm.

7.3. Development of the Objective Function

Nearly all algorithms for solving the minimum-area placement problem are based on simulated annealing. It is, however, well known from several other optimization problems that simulated annealing has slow convergence toward good solutions—hence, more greedy algorithms have been proposed in the literature, e.g., tabu search, noising algorithms, and guided local search. When developing the present algorithm, several of the mentioned metaheuristics were implemented using SEMINORM-SP, but the results were disappointing. To explain this behavior, the objective function was plotted in Figure 10 as a function of the iterations performed in a simulated annealing

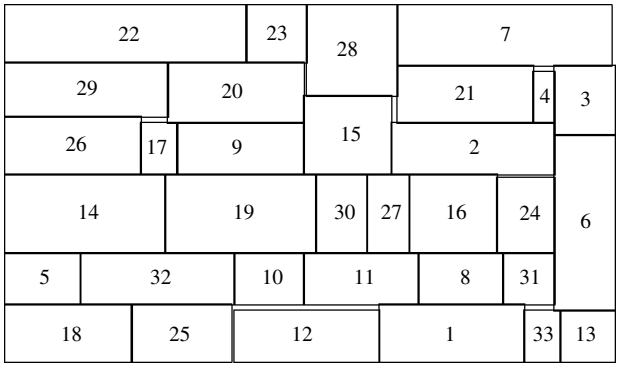


Figure 8 Best Found Solution ami33

Note. Area: 1.167964 mm^2. Time: 1,359 s. Size: 1,414 × 826. Rotations allowed.

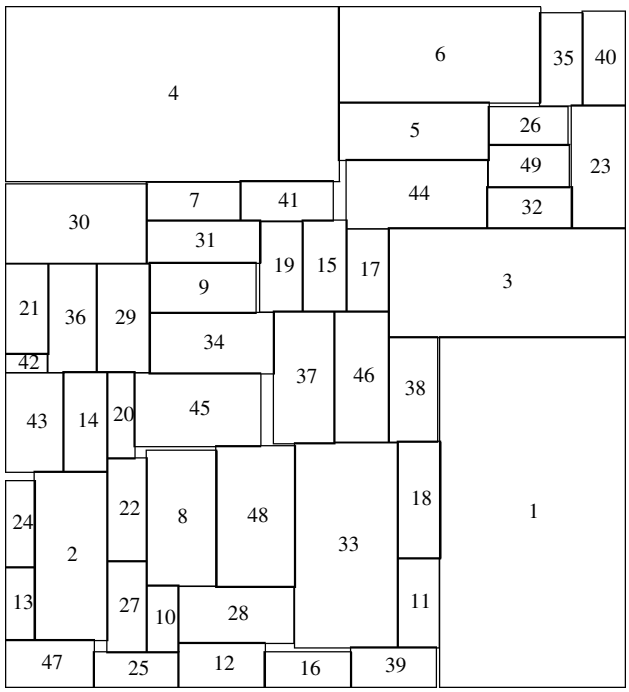


Figure 9 Best Found Solution ami49

Note. Area: 35.993636 mm^2. Time: 3,004 s. Size: 5,726 × 6,286. Rotations allowed.

algorithm. Although it is the outcome of a specific run of simulated annealing, it illustrates the topology of the objective function. The objective function is very unstable—minor changes to the sequence pair may result in large fluctuations in the objective value. Hence, using modern metaheuristics that investigate every local minimum very carefully makes no sense as the objective function is not smooth.

Figure 10 also shows that the SEMINORM-SP algorithm can find very high-quality solutions in split seconds. Indeed the majority of the solution time is used for improving the solution value by a few percent.

7.4. Bin-Packing Instances

To show the wide applicability of the presented algorithm we have tested it for a number of two-dimensional bin-packing instances provided by Berkey and Wang (1987) and Martello and Vigo (1998). These instances consist of ten classes of problems. In each problem class there are 50 instances: ten with 20 rectangles, ten with 40 rectangles, ten with 60 rectangles, ten with 80 rectangles and ten with 100 rectangles. The rectangles may not be rotated. Problem classes I–VI have been proposed by Berkey and Wang (1987) while the last four classes have been proposed by Martello and Vigo (1998). The characteristics of the classes are summarized in Table 4. In our setting we want to find the smallest rectangle enclosing all rectangles where the bin-packing problem deals with rectangular bins of fixed size $W \times H$. Due to this different setting, instance classes I, III, and V are identical to instance classes II, IV, and VI, respectively. Hence, we will not consider the latter instances.

In Table 5 we report the area of the enclosing rectangle as the average of the ten instances considered for each problem size. Figure 11 shows some

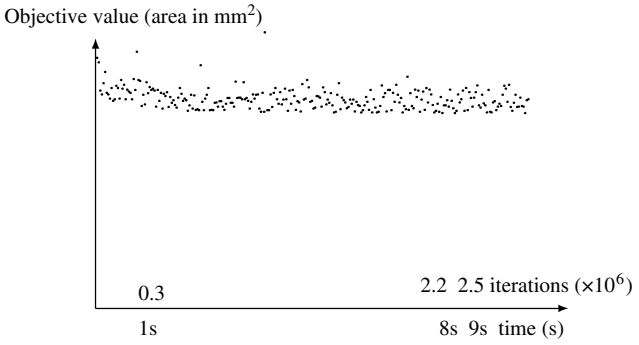


Figure 10 Development of the Objective Function as a Function of the Number of Iterations

Table 4 Instance Classes Considered

Item type 1	w_r uniformly random in $[\frac{2}{3}W, W]$, h_r uniformly random in $[1, \frac{1}{2}H]$
Item type 2	w_r uniformly random in $[1, \frac{1}{2}W]$, h_r uniformly random in $[\frac{2}{3}H, H]$
Item type 3	w_r uniformly random in $[\frac{1}{2}W, W]$, h_r uniformly random in $[\frac{1}{2}H, H]$
Item type 4	w_r uniformly random in $[1, \frac{1}{2}W]$, h_r uniformly random in $[1, \frac{1}{2}H]$
Class I	w_r and h_r uniformly random in $[1, 10]$, $W = H = 10$
Class II	w_r and h_r uniformly random in $[1, 10]$, $W = H = 30$
Class III	w_r and h_r uniformly random in $[1, 35]$, $W = H = 40$
Class IV	w_r and h_r uniformly random in $[1, 35]$, $W = H = 100$
Class V	w_r and h_r uniformly random in $[1, 100]$, $W = H = 100$
Class VI	w_r and h_r uniformly random in $[1, 100]$, $W = H = 300$
Class VII	item type 1 with probability 70%, type 2, 3, 4 with probability 10% each. $W = H = 100$
Class VIII	item type 2 with probability 70%, type 1, 3, 4 with probability 10% each. $W = H = 100$
Class IX	item type 3 with probability 70%, type 1, 2, 4 with probability 10% each. $W = H = 100$
Class X	item type 4 with probability 70%, type 1, 2, 3 with probability 10% each. $W = H = 100$

Table 5 Bin-Packing Instances: Area Used

	Class						
n	I	III	V	VII	VIII	IX	X
20	594	6,349	50,011	43,885	45,068	92,111	34,000
40	1,193	12,790	100,656	94,408	95,315	181,362	65,408
60	1,878	19,939	157,059	138,730	140,540	280,232	93,122
80	2,616	27,680	217,812	199,975	197,249	376,285	121,192
100	3,195	33,509	263,171	241,372	244,314	459,167	152,882

Note. Average of ten instances. Each instance is best of ten runs. Rotations not allowed.

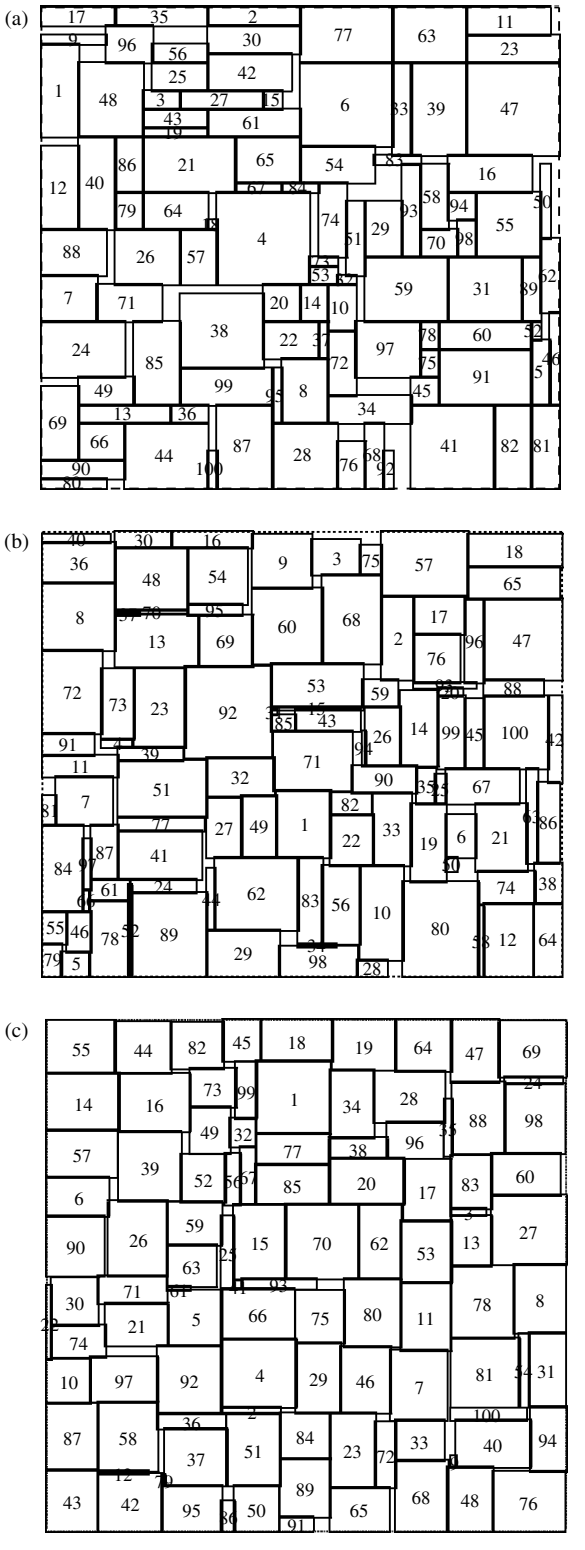


Figure 11 **Some Selected Bin-Packing Solutions with $n = 100$**

selected solutions: (a) Class I instance with area 2,912, size 56×52, and waste 6.1%; (b) Class III instance with area 31,845, size 193×165, and waste 4.1%; (c) Class IX instance with area 467,831, size 689×679, and waste 3.0%.

Table 6 **Bin-Packing Instances: Wasted Space in Percent**

	Class						
n	I	III	V	VII	VIII	IX	X
20	2.58	4.04	4.38	3.40	3.75	3.34	3.83
40	2.81	3.59	3.70	3.16	3.39	3.17	3.35
60	4.55	3.50	3.59	3.35	3.22	3.01	3.42
80	5.17	3.67	3.72	3.25	3.24	2.94	3.32
100	6.18	4.10	3.90	3.29	3.33	3.03	3.83

Note. Average of ten instances. Each instance is best of ten runs. Rotations not allowed.

Table 6 reports the wasted space in percentages. It is seen that the wasted space is between 2.5% and 6% for Class I instances. This is slightly more than for the other instance types considered, but can be explained by the fact that all rectangles have quite small integer dimensions. Thus, each rectangle can only be placed on a small number of grid positions. This can also be seen by comparing Figure 11(a) and (b).

For all other instance classes in Table 6 the wasted space is stable between 3% and 4%. As a whole the wasted space is about twice as large as for the other problem types considered in this paper, which may be explained by the fact that rotations are not allowed for bin-packing instances.

Finally, Table 7 shows the running times of the algorithm per run. The running times are very moderate—around half a minute for instances with 100 rectangles.

7.5. Packing of Squares

Finally, we consider instances presented by Korf (2004). For a given value of n the instance consists of squares of size $1 \times 1, 2 \times 2, \ldots, n \times n$. The objective is to pack the squares into a rectangle of minimum area. Korf solved the problems to optimality for n up to 25 using a branch-and-bound algorithm with additional dominance tests.

In Table 8 we report the used area, the wasted area in percent, and the solution time for packing n squares using the presented heuristic. For symmetry reasons we do not allow rotations in the solution process. Short runs use $100{,}000n$ iterations, while long runs use $1{,}000{,}000n$ iterations. For the short runs the

Table 7 **Bin-Packing Instances: Solution Times Per Run**

	Class						
n	I	III	V	VII	VIII	IX	X
20	1.97	1.95	1.93	1.83	1.88	1.93	1.94
40	6.28	6.30	6.15	6.14	6.06	6.21	6.28
60	12.95	12.94	13.08	12.98	12.79	13.04	13.20
80	23.06	23.07	22.87	22.66	22.29	22.73	23.34
100	36.20	35.62	35.92	35.35	35.07	35.54	36.60

Note. Average of ten instances. Each instance is best of ten runs. Rotations not allowed.

Table 8 Packing of Squares. All Values are Best Out of Ten Runs. Rotations Not Allowed

n	10	15	20	25	30	40	50	60	70	80	90	100
Optimal												
Area	405	1,265	2,890	5,547	—	—	—	—	—	—	—	—
Waste (%)	4.94	1.98	0.69	0.40	—	—	—	—	—	—	—	—
Short												
Area	405	1,295	2,970	5,694	9,752	22,752	44,099	75,915	120,056	178,893	254,214	347,420
Waste (%)	4.94	4.25	3.37	2.97	3.05	2.69	2.66	2.77	2.72	2.80	2.81	2.61
Time (s)	0.69	1.26	1.97	2.93	3.92	6.56	9.85	13.52	18.27	23.92	30.32	37.01
Long												
Area	405	1,275	2,968	5,655	9,717	22,578	43,575	75,137	118,703	176,808	251,460	343,371
Waste (%)	4.94	2.75	3.30	2.30	2.70	1.94	1.49	1.77	1.61	1.66	1.75	1.46
Time (s)	6.95	12.58	19.80	28.88	39.43	65.59	97.80	134.89	180.08	236.24	302.13	372.79

found solutions waste between 2.5% and 3% of the area, while for the long runs the waste is between 1.5% and 2%. Figure 12 depicts the best found solution for $n = 100$.

8. Conclusion

A new transformation of sequence pair to placement has been presented running in $O(n \log \log n)$ time, which matches the best running times in the literature. As opposed to previous sequence pair transformations, the output of SEMINORM-SP is semi-normalized placements. SP and SEMINORM-SP generate the same placement for a minimum-area solution. Thus, we may use the latter transformation in a local search algorithm, taking advantage of the fact that all placements considered will use no more area than the original transformation.

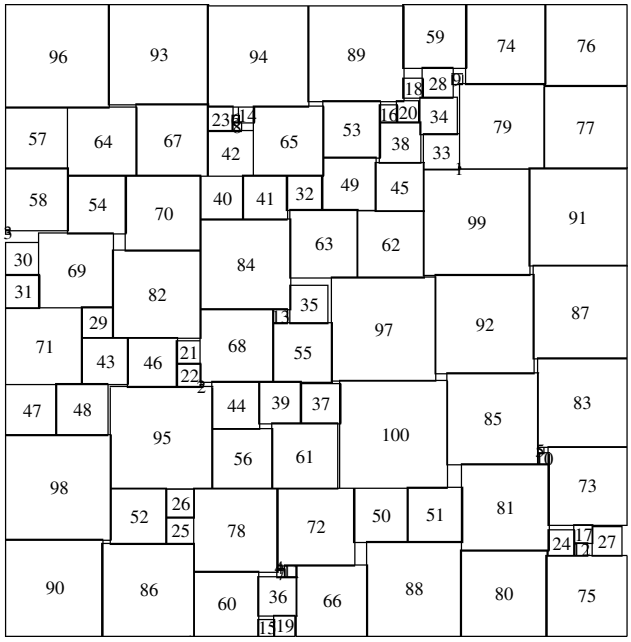


Figure 12 Packing of Squares, $n = 100$. Area: 343,371, Time: 372 s, Size 581×591. Rotations Not Allowed

The semi-normalized interpretation of sequence pair seems very attractive compared to other abstract representations like O-tree, B*-tree, and corner block list. The time used for transforming a sequence pair to a placement is almost linear, the placement is automatically normalized, and any minimum-area placement can be represented by a sequence pair. None of the other representations possess all these properties.

The computational results show that SEMINORM-SP can be applied to a large variety of packing problems, finding solutions with a very moderate space waste of a few percent.

Acknowledgments

The author thanks Jens Egeblad, Oluf Færø, and Martin Zachariasen for constructive discussions. The comments of the two anonymous referees and the associate editor are greatly appreciated.

References

Berkey, J. O., P. Y. Wang. 1987. Two dimensional finite bin packing algorithms. *J. Oper. Res. Society* **38** 423–429.

Chang, Y. C., Y. W. Chang, G. M. Wu, S. W. Wu. 2000. B*-trees: A new representation for non-slicing floorplans. *Proc. 37th ACM/IEEE Conf. Design Automation, Los Angeles, CA*, 458–463.

Christofides, N., C. Whitlock. 1977. An algorithm for two-dimensional cutting problems. *Oper. Res.* **25** 30–44.

Guo, P. N., C. K. Cheng, T. Yoshimura. 1999. An O-tree representation of non-slicing floorplans and its applications. *Proc. 36th ACM/IEEE Conf. Design Automation, New Orleans, LA*, 268–273.

Hong, X., G. Huang, Y. Cai, J. Gu, S. Dong, C.-K. Cheng, J. Gu. 2000. Corner block list: An effective and efficient topological representation of non-slicing floorplan. *Internat. Conf. Computer-Aided Design, San Jose, CA*, 8–12.

Iori, M., S. Martello, M. Monaci. 2003. Metaheuristic algorithms for the strip packing problem. P. M. Pardalos, V. Korotkikh, eds. *Optimization and Industry: New Frontiers*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 159–179.

Korf, R. E. 2004. Optimal rectangle packing: New results. *14th Internat. Conf. Automated Planning and Scheduling, Whistler, British Columbia, Canada*.

Martello, S., D. Vigo. 1998. Exact solution of the two-dimensional finite bin packing problem. *Management Sci.* **44** 388–399.

Martello, S., M. Monaci, D. Vigo. 2003. An exact approach to the strip-packing problem. *INFORMS J. Comput.* **15** 310–319.

Martello, S., D. Pisinger, D. Vigo. 2000. The three-dimensional bin packing problem. *Oper. Res.* **48** 256–267.

MCNC. 2001. Floorplan benchmark tests, http://www.cse.ucsc.edu/research/surf/GSRC/MCNCbench.html.

Murata, H., K. Fujiyoshi, S. Nakatake, Y. Kajitani. 1996. VLSI module placement based on rectangle-packing by the sequence pair. *IEEE Trans. Comput. Aided Design Integrated Circuits Systems* **15** 1518–1524.

Pang, Y., C. K. Cheng, T. Yoshimura. 2000. An enhanced perturbing algorithm for floorplan design using the O-tree representation. *Proc. Internat. Sympos. Physical Design, San Diego, CA,* 168–173.

Takahashi, T. 1996. An algorithm for finding a maximum-weight decreasing sequence in a permutation, motivated by rectangle packing problem. IEICE Technical Report VLD96, 201, 31–35, Institute of Electronics, Information and Communication Engineers, Tokyo, Japan.

Tang, X., D. F. Wong. 2001. FAST-SP: A fast algorithm for block placement based on sequence pair. *Proc. Asia and South Pacific Design Automation Conf., Yokohama, Japan,* 521–526.

Tang, X., R. Tian, D. F. Wong. 2000. Fast evaluation of sequence pair in block placement by longest common subsequence computation. *Proc. Design, Automation and Test in Europe (DATE2000), Paris, France,* 106–111.

van Emde Boas, P. 1977. Preserving order in a forest in less than logarithmic time and linear space. *Inform. Processing Lett.* **6** 80–82.