An efficient placement method for large Standard-Cell and Sea-of-Gates designs

H.J. Kappen, F.M.J. de Bont,
Philips Research Laboratories,
P.O. Box 80000, 5600 JA Eindhoven, the Netherlands

Abstract
A fast placement algorithm is presented for large Standard-Cell and Sea-of-Gates placement problems. The time complexity of the algorithm is \( O(n \log^2 n) \). In comparison with a state of the art conventional placement and routing tool it yields more than 10% smaller layouts with significantly better timing characteristics. The method underlying this algorithm uses a quadratic cost function of the wire length and has a wider applicability than existing methods. We expect that with this algorithm high quality placements up to 100000 cells can be obtained.

Keywords: Standard Cell placement, optimization, Quadratic Assignment Problem, recursive partitioning.

1 Introduction
During the last few years the demand for algorithms that can yield good layouts for increasingly larger chips has steadily increased. In layout styles, such as Sea-of-Gates and Standard-Cell, the number of basic cells to be placed can easily amount to several tens or hundreds of thousands.

The quality of a Standard-Cell or Sea-of-Gates layout is characterized by its timing characteristics and the total area occupied by the interconnect after routing. The problem is to find a placement of the cells such that, after routing, both the area of the chip is minimized, and the timing specifications are met.

The approach that we will follow in this paper, is the use of a cost function that sums the squared wire lengths between the cells, which we will call the quadratic cost function [2]. Minimizing a quadratic cost function has the effect of penalizing long wires more than for instance using a min-cut criterion [8]. This results in better layouts because both timing and area depend mainly on the long wires in a placement: the long wires usually determine the overall timing behaviour, and they largely determine the number of tracks of a routing channel. The short wires can often be routed in residual channel space without enlarging the chip area. Our numerical results will illustrate this point.

The method that we introduce in this paper, which is called the Tangency method, is based on recursive partitioning and is similar to the method presented by the Kuh et al. [3]. Each partition is derived from the solution of a set of linear equations. Kuh's method only works, however, when initially a minimal number of cells is given fixed positions on the boundary of the chip. Our method does not suffer from this restriction: problem instances without external connections, or with only a limited number of them, can be handled. This can be done at the cost of having to solve a set of \( n + 1 \) linear equations instead of \( n \), where \( n \) is the number of cells in the sub-problem that has to be partitioned.

We will restrict ourselves in this paper to placement onto a rectangular set of grid points. This should not be viewed as a limitation of our method. Extension to non-grid placements is straightforward. The resulting problem is called the Quadratic Assignment Problem and is introduced in Section 2. In Section 3.1 we introduce the Tangency method. In Section 4 we present some numerical results for Standard-Cell. For a full length discussion see [4].

2 The Quadratic Assignment Problem
Let there be \( n \) cells, named 1, \ldots, \( n \), for which a placement has to be calculated, and \( m \) cells, named \( n + 1, \ldots, n + m \), whose positions are known. These cells will be referred to as the free and the border cells, respectively. Let the positions of the cells in the two-dimensional real plane be given by two real \( n + m \)-dimensional vectors \( \mathbf{X} \) and \( \mathbf{Y} \), such that \((X_i, Y_i)\) denotes the position of cell \( i \). The unknown positions of the free cells are denoted by \((X_i, Y_i) = (x_i, y_i), 1 \leq i \leq n \) and the positions of the border cells by \((X_i, Y_i) = (\hat{x}_i, \hat{y}_i), n + 1 \leq i \leq n + m \). A pair \((x, y)\) is called a configuration of the cells.

The interconnect between the cells is represented in terms of a graph: the nodes and the edges denoting the cells and the wires, respectively. In general, VLSI circuits contain \( k \) point connections between cells, called nets, and not just two point connections. In addition, it is sometimes required to 'emphasize' so called critical nets by assigning weights to them: non-critical nets have a weight equal to one and critical nets have a weight larger than one. A net with weight \( s \) that connects \( k \) cells is represented in the graph as a clique on those cells, all edges having a weight \( 2s/k \). In this way the total clique 'weights' as much as the original net, i.e. \((k - 1)s\). The edges of the graph are stored in a connectivity matrix \( \mathbf{C} \), which for most VLSI placement problems may assumed to be...
Let the component of \( C \) be denoted by \( C_{ij} \in \mathbb{R}, i, j = 1, \ldots, n + m \). Then the cost of a configuration of \( n \) free cells and \( m \) border cells is given by the function \( w_{n,m} : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R} \), so that

\[
    w_{n,m}(x, y) = \frac{1}{2} \sum_{i=1}^{n+m} \sum_{j=1}^{n+m} C_{ij}[(X_i - X_j)^2 + (Y_i - Y_j)^2].
\]

Let

\[
    d_i = \sum_{j=1}^{n+m} C_{ij}, \quad i = 1, \ldots, n + m
\]

and \( \delta_{ij} \) be the Kronecker delta (\( \delta_{ij} = 1 \) if \( i = j \); \( \delta_{ij} = 0 \) otherwise). Then we can rewrite Equation (1) as

\[
    w_{n,m}(x, y) = x^T A x + b^T x + c_x + y^T A y + b_y^T y + c_y
\]

\[
    \equiv w_{n,m}(x) + w_{n,m}(y).
\]

where \( A \) denotes a real symmetric \( n \times n \) matrix with components

\[
    A_{ij} = \delta_{ij} - C_{ij}, \quad i, j = 1, \ldots, n.
\]

The components of the vector \( b_x \) are

\[
    b_{xi} = -2 \sum_{j=1}^{n+m} C_{ij} x_j, \quad i = 1, \ldots, n,
\]

and the constant

\[
    c_x = \sum_{i=1}^{n+m} \Delta i - C_{ij} x_j x_j.
\]

The vector \( b_y \) and \( c_y \) are defined analogously. Let the chip be represented in the real two-dimensional plane by a rectangular orthogonal grid of \( n \) points \( G_{r,s,t,u} = \{(x, y) \in \mathbb{R}^2 | x \in \{t + 1, \ldots, t + r\}, y \in \{u + 1, \ldots, u + s\}\} \) with \( r, s, t, u \in \mathbb{Z}, r, s \geq 1 \) and \( t + r = n \) and \( s + u = n \). Let the set of free cells be denoted by \( S_n = \{1, \ldots, n\} \).

Let \( L_{r,s,t,u} \) denote the set of placements on \( G_{r,s,t,u} \), i.e. those configurations of the free cells for which each grid point is occupied by a cell, and no two cells are placed at the same grid point:

\[
    L_{r,s,t,u} = \{(x, y) | (x_i, y_i) \in G_{r,s,t,u}, i = j \Rightarrow x_i = x_j \land y_i = y_j\}.
\]

The Quadratic Assignment Problem is defined as

\[
    (G_{r,s,t,u}, S_n, w_{n,m}) : \text{find } (x', y') \in L_{r,s,t,u} \text{ such that } w_{n,m}(x', y') = \min_{(x,y) \in L_{r,s,t,u}} w_{n,m}(x, y).
\]

This problem is known to be NP-complete [5].

### 3 Recursive Partitioning

Our task is to develop a fast heuristic that finds a near optimal solution to the Quadratic Assignment Problem. Our strategy will be recursive partitioning on \( (G_{r,s,t,u}, S_n, w_{n,m}) \): we will show how the Quadratic Assignment Problem is approximated by a series of Partitioning Problems. Kuhl’s method and the Tangency method are approximate methods for solving the Partitioning Problem and are discussed in the following subsection.

Let the grid that we want to partition be given by \( G_{r,s,t,u} \) and suppose we want to perform a partitioning step in the \( x \)-direction. We define \( r_1 = \lfloor \frac{r}{2} \rfloor \), \( r_2 = r - r_1 \), \( n_1 = r_1 s \) and \( n_2 = r_2 s \). Then we can write \( G_{r,s,t,u} \) as the union of two smaller grids, each roughly half the size of the original grid:

\[
    G_{r,s,t,u} = G_{r_1,s,t,u} \cup G_{r_2,s,t,u}.
\]

A \((x)\)-partition of the grid \( G_{r,s,t,u} \) is a configuration \( x \) with

\[
    x(i) = t + \frac{r_1 - 1}{2}, \quad i = 1, \ldots, n_1, \quad x(i) = t + r_2 + \frac{r_1 - 1}{2}, \quad i = n_1 + 1, \ldots, n.
\]

The Partitioning Problem is to find the partition of the grid \( G_{r,s,t,u} \) that minimizes \( w_{n_1,m}(x) \). The Partitioning Problem is a special case of the Quadratic Assignment Problem and is NP-complete [5]. Note that for a partitioning step in the \( x \)-direction only the \( x \) coordinates of the cells are relevant.

Let us assume for the time being that we have found a near optimal solution \( x \) of the Partitioning Problem. Then we can use this partition to define two new placement problems:

\[
    (G_{r_1,s,t,u}, S_{n_1}, w_{n_1,m}) \rightarrow (G_{r_1,s,t+1,u}, S_{n_1}, w_{n_1,m+1})
\]

\[
    (G_{r_2,s,t+1,u}, S_{n_2}, w_{n_2,m+1})
\]

Note that for instance the sub-problem \( (G_{r_1,s,t+1,u}, S_{n_1}, w_{n_1,m+1}) \) has \( n_1 \) extra border cells, which are all placed at the centre of gravity of the other sub grid, i.e. \( (t + r_2 + \frac{r_1 - 1}{2}, u + \frac{u + s - 1}{2}) \).

A partitioning step in the \( y \)-direction can be defined analogously.

A partitioning step is illustrated in Figure 1.

The cost function whose minimum is sought during a partitioning step in the \( x \) or \( y \)-direction, is the quadratic cost function \( w_{n_1,m}(x) \) or \( w_{n_2,m}(y) \) over all cells, including the border cells. Among the border cells there are cells which are located at the centres of areas to which cells have been assigned during previous partitioning steps (the solid dots in Figure 1). Those cells have been placed, temporarily, at these centres, until a later partitioning step separates them. If bonding pads are present, there are additional border cells around the border
of the chip.

After \( k \) partitioning steps, let the grids be given by \( G_{r,s,t,u} \), \( i = 1,\ldots,2^k \). Define the size of a grid \( G_{r,s,t,u} \) as \( |G_{r,s,t,u}| = \max(r,s) \). The method uses the following criterion to decide whether or not at all each of the \( G_{r,s,t,u} \) should be partitioned in the \( x \) or in the \( y \)-direction. Let \( l \) be an integer: if \( |G_{r,s,t,u}| > l \) and \( r \geq s \) then partition in the \( x \)-direction, if \( |G_{r,s,t,u}| > l \) and \( r < s \) then partition in the \( y \)-direction. The algorithm terminates when \( k \) is such that \( |G_{r,s,t,u}| \leq l, i = 1,\ldots,2^k \). We will use \( l = 1 \), but \( l > 1 \) has also been used successfully [6].

So far we have shown how the recursive partitioning scheme works and how the solution of each partitioning problem can be used to split a placement problem into two placement problems of smaller size. In the rest of this section we will discuss Kuh's method and the Tangency method. These are methods to obtain a near optimal solution to the Partitioning Problem.

### 3.1 The Tangency Method

The configuration \( x^0 \in \mathbb{R}^n \) that minimizes \( w_{(n)} \), is easily derived by differentiating Equation (2) with respect to \( x \), giving

\[
2Ax^0 + b_x = 0. \tag{3}
\]

Figure 2: The solution \((x^0, y^0)\) that minimizes Equation (3). The border cells are drawn as closed dots and the free cells as open dots. The solution places the free cells within the convex hull of the border cells.

This minimal configuration places the cells inside the convex hull of the border cells (see Figure 2).

A permutation \( \pi \in \Pi_n \) is called an ordering of a configuration \( x \) if \( x_{\pi(1)} \leq \cdots \leq x_{\pi(n)} \).

The cell positions of the optimal configuration can be changed by changing the positions of the border cells. In Kuh's method border cells are placed so that they will 'pull' the optimal configuration \( x^0 \) in such a way that roughly as many cells are placed in the left as in the right sub grid. A partition \( x \) with the same ordering as \( x^0 \) is easily constructed. If \( x^0 \) 'does not differ too much' from a partition, \( x \) can be expected to be a near optimal solution of the Partitioning problem. This method works well, but only on problem instances with relatively many border cells. For practical applications this is rather unsatisfying.

The Tangency method solves this problem and is applicable to any problem instance (whether without external connections or with just a few). This is how it works. Let us call a configuration \( x \) balanced with respect to a the grid \( G_{r,s,t,u} \) when the average value of the cell positions is the same as that of a partition,

\[
\sum_{i=1}^n x_i = n(t + r + 1)/2, \tag{4}
\]

or, in vector notation,

\[(x - a)^T a = 0, \quad a = (t + r + 1)/2 \cdot (1, \ldots, 1). \tag{5}\]

The configurations that satisfy Equation (5) lie in an \((n-1)\)-dimensional subspace \( V \subseteq \mathbb{R}^n \).

We will be more specific than above by making the assumption that a balanced optimal configuration \( x^0 \) 'does not differ too much' from a partition, so that its ordering can be used to construct a near optimal partition. The strength of the Tangency method is that balancedness in the form of Equation (4) can be enforced without putting any requirements on the number and positions of the border cells.

As we saw above, for a partitioning step in the \( x \)-direction only the function \( w_{(n)} \) is relevant. For all \( c \in \mathbb{R} \), the set of points

\[ E_c = \{ x \in \mathbb{R}^n | w_{(n)}(x) = c \} \]

describes a hyper-ellipse in \( \mathbb{R}^n \). The centre of the ellipse is given by the solution \( x^0 \) of Equation (3), and minimizes the quadratic cost function \( w_{(n)}(x) \). Kuh's idea was to construct a partition from \( x^0 \). The method that we propose is to construct a partition from the point \( x^1 \in V \) that minimizes the quadratic cost function in \( V \). Let \( c \) be this minimal cost. The vector \( x^1 \) is then the point of tangency of \( E_c \) and \( V \), and \( x^1 \) is balanced by construction.

The vectors normal to \( V \) and \( E_c \) at a point \( x \in \mathbb{R}^n \) are respectively given by

\[ n_V = a, \quad n_E = 2Ax_0 + b_x. \]

So the condition for tangency is to find \( \lambda \in \mathbb{R} \) and \( x^0 \in \mathbb{R}^n \), so that

\[ 2Ax^0 + b_x = \lambda a \quad \text{and} \quad (x^0 - a)^T a = 0. \tag{6} \]

Equations (6) describe \( n + 1 \) linear equations with \( n + 1 \) unknowns, \( x^0_1, \ldots, x^0_n \) and \( \lambda \).

Let \( \pi \in \Pi_n \) be an ordering of \( x^1 \). Then the requirement

\[ x_{\pi(1)} \leq \cdots \leq x_{\pi(n)} \]

and \( x \) a partition of \( G_{r,s,t,u} \) defines \( x \) uniquely. That is, the partition \( x \) is chosen such that its ordering is the same as the ordering of \( x^1 \). In Figure (3), the Tangency method is illustrated in a particularly simple situation.

The above scheme does not work when external connections are absent, i.e. \( m = 0 \). In this case the permutation \( \pi \) is calculated from the eigenvector \( v_2 \) [4].
ever, when the resulting placement is used as a Standard-Cell placement in row length. This part has worse than or Sea-of-Gates layout, the rows will differ in length. An iterative improvement algorithm was added to reduce this variability.

When the differences in cell sizes are ignored and the nets are converted to cliques in a graph, the problem reduces to the Quadratic Assignment Problem (QAP). When the border cells are distributed so that Equation (4) holds exactly, the Tangency method and Kuh’s method yield the same result, i.e., \( x^1 = x^0 \). For all problem instances for which Kuh’s method works, the border cells will be distributed such that for each partitioning step Equation (4) will be approximately true. Therefore we expect both algorithms to perform equally well on these problem instances. Since the set of problem instances for which Kuh’s method is suitable is a sub-set of the set of problem instances for which the Tangency method works, and because a similar performance can be expected on the intersection of these sets, a numerical comparison between the Tangency method and Kuh’s method should therefore be seen as a crude estimate of the success of the method.

The results that are presented in this section were obtained with the Tangency method routed with the SCII router. SCII is a state of the art commercial placement and routing tool of Silvaco. The results were compared with layouts obtained with the SCII placer and router. The SCII placer is based on a recursive min-cut strategy of the type discussed in [8].

The numerical results are presented in Table 1. Here \( n \) denotes the number of cells. Under ‘SCII’ resp. ‘Tangency’ some characteristics of the layouts, corresponding to SCII resp. Tangency placements are summarized. Under ‘area’, the areas of the bounding box of the rows of cells are given in mm\(^2\), i.e., bonding pads and input/output wires are excluded. Under ‘load’, the average load and standard deviation over all the nets is given in 0.1 pf. Under ‘cpu’, the cpu-times on Apollo DN4000 for placement only are given in seconds.

### Table 1: Comparison of the Tangency placement algorithm and the SCII placement algorithm. Both placements are routed with the SCII router.

<table>
<thead>
<tr>
<th>n</th>
<th>area</th>
<th>load</th>
<th>cpu</th>
<th>area</th>
<th>load</th>
<th>cpu</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.26</td>
<td>0.91±1.12</td>
<td>75</td>
<td>0.27</td>
<td>0.99±0.79</td>
<td>50</td>
</tr>
<tr>
<td>266</td>
<td>0.75</td>
<td>1.02±0.77</td>
<td>165</td>
<td>0.74</td>
<td>0.74±0.60</td>
<td>110</td>
</tr>
<tr>
<td>427</td>
<td>1.46</td>
<td>1.42±3.05</td>
<td>570</td>
<td>1.54</td>
<td>1.58±2.72</td>
<td>135</td>
</tr>
<tr>
<td>428</td>
<td>1.42</td>
<td>1.08±1.66</td>
<td>420</td>
<td>1.39</td>
<td>1.08±1.41</td>
<td>125</td>
</tr>
<tr>
<td>513</td>
<td>1.77</td>
<td>1.43±2.05</td>
<td>660</td>
<td>1.75</td>
<td>1.42±2.77</td>
<td>225</td>
</tr>
<tr>
<td>672</td>
<td>2.54</td>
<td>1.73±4.10</td>
<td>1500</td>
<td>2.42</td>
<td>1.69±3.67</td>
<td>270</td>
</tr>
<tr>
<td>839</td>
<td>3.02</td>
<td>1.21±2.23</td>
<td>1200</td>
<td>2.97</td>
<td>1.22±1.50</td>
<td>280</td>
</tr>
<tr>
<td>1300</td>
<td>6.54</td>
<td>2.39±5.95</td>
<td>4500</td>
<td>6.22</td>
<td>2.29±5.25</td>
<td>340</td>
</tr>
<tr>
<td>1323</td>
<td>5.05</td>
<td>1.61±2.14</td>
<td>2400</td>
<td>4.83</td>
<td>1.59±1.19</td>
<td>695</td>
</tr>
<tr>
<td>1862</td>
<td>7.53</td>
<td>1.59±2.01</td>
<td>3450</td>
<td>6.86</td>
<td>1.64±1.04</td>
<td>1055</td>
</tr>
<tr>
<td>3239</td>
<td>23.67</td>
<td>2.77±5.54</td>
<td>9600</td>
<td>9.82</td>
<td>2.75±2.94</td>
<td>1920</td>
</tr>
<tr>
<td>3329</td>
<td>16.50</td>
<td>2.05±3.25</td>
<td>10680</td>
<td>13.23</td>
<td>1.85±1.50</td>
<td>1950</td>
</tr>
<tr>
<td>4754</td>
<td>21.93</td>
<td>1.82±6.33</td>
<td>7740</td>
<td>20.98</td>
<td>1.87±4.44</td>
<td>3750</td>
</tr>
<tr>
<td>5996</td>
<td>36.26</td>
<td>2.68±4.16</td>
<td>33480</td>
<td>30.15</td>
<td>2.37±2.05</td>
<td>4250</td>
</tr>
<tr>
<td>11467</td>
<td></td>
<td></td>
<td></td>
<td>90000</td>
<td></td>
<td>9000</td>
</tr>
</tbody>
</table>

4 Numerical Results

We did a preliminary study of the performance of the Tangency method for a number of Standard-Cell placement problems and one Sea-of-Gates placement problem. The cells from our libraries all have the same height, but differ in width. When the differences in cell sizes are ignored and the nets are converted to cliques in a graph, the problem reduces to the Quadratic Assignment Problem as defined in Section 2. However, when the resulting placement is used as a Standard-Cell or Sea-of-Gates layout, the rows will differ in length. An iterative improvement algorithm was added to reduce this variation in row length. This part has worse than \( O(n^2) \) time complexity. A time bound was introduced so that the iterative improvement part could use maximally as much time as the Tangency method. Of course, the proper way to deal with this is to modify the Tangency method so that variable cell sizes are taken into account (as is done in Kuh’s method [3]). This modification could be easily implemented but has not been done so far. The results that are presented in this section should therefore be seen as a crude estimate of the success of the method.

The Standard Cell placements obtained with the Tangency method were routed with the SCII router. SCII is a state of the art commercial placement and routing tool of Silvaco. The results were compared with layouts obtained with the SCII placer and router. The SCII placer is based on a recursive min-cut strategy of the type discussed in [8].

The numerical results are presented in Table 1. Here \( n \) denotes the number of cells. Under 'SCII' resp. 'Tangency' some characteristics of the layouts, corresponding to SCII resp. Tangency placements are summarized. Under 'area', the areas of the bounding box of the rows of cells are given in mm\(^2\), i.e., bonding pads and input/output wires are excluded. Under 'load', the average load and standard deviation over all the nets is given in 0.1 pf. Under 'cpu', the cpu-times on Apollo DN4000 for placement only are given in seconds.
The average load of the nets is roughly the same for the Tangency method and SCII. However, the standard deviation of the loads and the maximum load are much smaller for the Tangency method (30% and 25% respectively). This indicates that the layouts obtained with the Tangency method contain fewer long nets, which means that their timing characteristics are better. This result is particularly interesting because a recent evaluation of several placement and routing tools at Philips Research (unpublished) selected SCII because of its good timing characteristics.

The CPU times that were measured for the Tangency method are in agreement with the expected $n \log^2 n$ behaviour.

We were not able to route our largest problem instance. The complexity of the SCII global router is roughly quadratic, and for large problems vast amounts of memory are required.

One experiment was performed for a triple metal Sea-of-Gates layout of a $10 \times 10$ multiplier. We compared the placement generated with the Tancell package with that obtained with Tangency. Both placements were routed with the GAS router [10]. For Sea-of-Gates, routing does not yield extra area, so that the comparison was made on average net length and standard deviation. The results are given in Table 2.

<table>
<thead>
<tr>
<th></th>
<th>av. net length</th>
<th>cpu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tancell</td>
<td>65.14 ± 93.77</td>
<td>'small weekend'</td>
</tr>
<tr>
<td>Tangency</td>
<td>65.85 ± 50.25</td>
<td>1365</td>
</tr>
</tbody>
</table>

Table 2: Comparison of the Tangency and Tancell placement algorithms for a Sea-of-Gates design of 1097 cells. Lengths are in gridpoints and cpu times are in seconds on Apollo DN4000. Both examples were routed with the same router which required approximately 5500 sec on Apollo DN4000.

Note that the average net length of the Tancell layout is slightly lower than of the Tangency layout. However, the large standard deviation of the Tancell layout indicates that it contains more long nets which is may lead to worse timing behaviour.

5 Conclusions

In this paper, the Tangency method has been introduced as a heuristic for approximating the Quadratic Assignment Problem. Generalization of the method to problems with cells of variable size is straightforward. The set of problem instances to which the method can be applied is a (large) super set of the set of problem instances to which Kuh's method applies. For problem instances to which both methods can be applied, both methods are equivalent.

For the Quadratic Assignment Problem without external connections, it was reported elsewhere [4] that the Tangency method is more effective and more efficient than Simulated Annealing.

For Standard-Cell placement problems it was illustrated that the use of the quadratic cost function leads to qualitatively better layouts and is very fast. From extrapolation we expect that high quality Standard-Cell or Sea-of-Gates placements of 100000 cells can be generated in approximately 33 hours on Apollo DN4000 (2 to 3 hours on DN10000). However, a more efficient routing tool would be needed for such large problems.

References

[9] LASO2, A set of Fortran IV subroutines for computing a few eigenvalues and eigenvectors of a large (sparse) symmetric matrix. Author D.S. Scott. Source available by sending message 'send index' to anl-mcs.ARPA!netlib