A macro placer algorithm for chip design

Endre Csóka, Attila Deák

September 8, 2015

Abstract

There is a set of rectangular macros with given dimensions, and there are wires connecting some pairs (or sets) of them. We have a placement area where these macros should be placed without overlaps in order to minimize the total length of wires. We present a heuristic algorithm which utilizes a special data structure for representing two dimensional step functions. This results in fast integral computation and function modification over rectangles. Our heuristics, especially our data structure for two-dimensional functions, may be useful in other applications, as well.

1 Introduction

A chip is composed of basic elements called cells, circuits, boxes or modules. They usually have a rectangular shape, contain several transistors and internal connections, and have (at least two) fixed pins. There is a netlist describing which pin should be connected to which other pins. The goal is to place the cells legally — without overlaps — in the chip area so as to minimize the total (weighted) length of the wires connecting the pins. This problem is also called the VLSI placement problem.

Finding the optimum is NP-hard, therefore, we present a heuristic algorithm based on primal-dual optimization inspired by the Hungarian Algorithm [6] for the minimum weight maximum matching problem. Namely, we use a cost function as dual function on the placement area, and we are looking for a placement minimizing the sum of the total netlength and the total costs of the areas covered by the macros. We try to find a non-negative cost function by which an almost optimal placement is legal even if we allow overlaps, and costs are counted with multiplicity. We will use an iterated algorithm on the space of primal-dual pairs based on the following two steps:

1. For every overlap, we increase the cost function under intersecting areas.
2. We try to find a better placement with respect to the new cost function.

We tried to focus on typical instances in practice, and we found that these have the following properties.

1. The placement area is not very large compared to the total size of the macros, but it is still easy to find a legal placement.
2. There are about a few hundreds of macros and every macro is contained in at most 10 nets.
3. Most of the nets connect two, sometimes three, and rarely more than three macros to each other.

Our method is optimized for such inputs.

This paper is organized as follows. In Section 2 we introduce some notations and give a formal definition of the macro placement problem. In Section 3 we describe the basic idea behind our algorithm and in Section 4 we present the algorithm. In Section 5 we describe some additional heuristics used in our placer.

2 The macro placement problem

Now we give a formal definition of the simplified macro placement problem. Let us denote by \( \mathcal{M} \) the set of macros. We assume that all pins of each macro are in the center of the macro. The place of a macro is identified with the
place of its center pin. For a macro $M$, denote its horizontal and vertical size by $size_x(M), size_y(M)$, respectively. For a macro $M$ at $(x, y)$, we denote the area occupied by $M$ by

$$S(M, (x, y)) = \left( x - \frac{size_x(M)}{2}, x + \frac{size_x(M)}{2} \right) \times \left( y - \frac{size_y(M)}{2}, y + \frac{size_y(M)}{2} \right).$$

A net $N$ is a subset of the macros that are connected.

**Definition 1.** A netlist is a pair $(\mathcal{M}, \mathcal{N})$ where $\mathcal{M}$ is a finite set of macros and $\mathcal{N} \subseteq \mathcal{P}(\mathcal{M})$ is a set of subsets of $\mathcal{M}$.

One can think of $\mathcal{N}$ as a hypergraph on $\mathcal{M}$, where each $N \in \mathcal{N}$ is a hyperedge. We assume $|N| \geq 2$ for each $N \in \mathcal{N}$.

**Definition 2.** The placement area is a rectangle denoted by $\mathcal{A}$. This contains a set of rectangular blockages $\mathcal{B}$ (where $\forall B \in \mathcal{B}, B \subset \mathcal{A}$). The sides of all rectangles are parallel to the axis.

A blockage is a part of the placement area where no macro can be placed.

**Definition 3.** A placement is a map $p : \mathcal{M} \mapsto \mathcal{A}$. The placement $p$ is legal if all of the followings hold.

- Every macro $M \in \mathcal{M}$ is placed in the placement area:
  $$S(M, p(M)) \subseteq \mathcal{A}.$$

- The places of any two macros $M, M' \in \mathcal{M}$ are disjoint:
  $$S(M, p(M)) \cap S(M', p(M')) = \emptyset.$$

- None of the macros $M \in \mathcal{M}$ are placed on a blockage $B \in \mathcal{B}$:
  $$S(M, p(M)) \cap B = \emptyset.$$

The macros have to be placed in the given orientation, these cannot be rotated. Let $(\mathcal{M}, \mathcal{N})$ be a netlist and $p$ a legal placement to the placement area $\mathcal{A}$ with blockages $\mathcal{B}$. Define $p$ on the set of nets $\mathcal{N}$ as follows. For $N = \{M_1, M_2, \ldots, M_k\} \in \mathcal{N}$, let $p(N) = (p(M_1), p(M_2), \ldots, p(M_k))$. We have a function $\mathcal{L} : \mathcal{A}^2 \cup \mathcal{A}^3 \cup \mathcal{A}^4 \cup \ldots \mapsto \mathbb{R}^+$ which evaluates the length of a net. $\mathcal{L}$ is also called the net (or netlength) model. One commonly used net model is the bounding-box model:

$$BB((x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)) = \max_i \{x_i\} - \min_i \{x_i\} + \max_i \{y_i\} - \min_i \{y_i\}$$

This is the half perimeter of the smallest rectangle with sides parallel to the axis, containing all pins of the macros contained in the net $N$.

**The Simplified Placement Problem:**

Given a netlist $(\mathcal{M}, \mathcal{N})$, a placement area $\mathcal{A}$, the set of blockages $\mathcal{B}$ and a net model $\mathcal{L}$, find a legal placement $p : \mathcal{M} \mapsto \mathcal{A}$ which minimizes the total netlength:

$$\sum_{N \in \mathcal{N}} \mathcal{L}(p(N)).$$

**3 Basic tools of the placer**

The initial problem is to place macros in the placement area (avoiding the blocked areas) so that the total netlength is minimal (or close to the minimum). As finding the optimum is NP hard, we present a heuristic algorithm with $O(\log(n) \log(m)s)$ running time, where the placement area is a discrete $n \times m$ grid and we run the algorithm for $s$ rounds.

We introduce our algorithm in several steps.
**Problem 1.** We have a set of macros $M$ and disjoint slots $A$. There is a cost function $c: M \times A \rightarrow \mathbb{R}^+$ which assigns costs to every possible macro-slot assignment. Find an injective assignment $p: M \mapsto A$ with minimum total cost

$$\sum_{M \in M} c(M, p(M)).$$

**Solution.** This scenario can be represented by a bipartite weighted graph. The two sets of points are $M$ and $A$ and the cost of an edge $(M, A)$ is $c(M, A)$. The task is to cover $M$ with a minimum cost maximum matching. This is a well known optimisation problem, and it is usually solved by primal-dual methods (e.g. the Hungarian Method [6]). However, we choose the following method instead, because we will generalize this in the later steps. We try to find a primal-dual solution by the following market simulation: if there is an area which is the best possible choice for at least two macros, then we increase the cost of that area.

**Problem 2.** For a macro $M \in M$ and a given netlist $N$, let $E(M)$ be the set of nets containing $M$:

$$E(M) = \{ N \in N \mid M \in N \},$$

and let $N(M)$ be the set of its neighbors:

$$N(M) = \{ M' \in M \mid \exists N \in E(M) : M' \in N \}.$$ 

Let $L(N)$ be the netlength model. Given a set of macros $M$, disjoint slots $A$ and netlist $N$, find an injective assignment $p: M \mapsto A$ which minimizes

$$\sum_{N \in N} L(p(N)).$$

**Solution.** We try to use the solution of Problem 1, where the cost of the placement of one macro is replaced by its marginal contribution to the total cost. Now, the cost of one macro depends on the placement of its neighbors, but there are not too many neighbors, therefore, we expect the cost function to change rather slowly. This allows us to use a variation of the above described market simulation:

1. Take an arbitrary macro-area assignment $p$.

2. For each macro $M \in M$, fix the other macros at their current position. For every $A \in A$, we get a placement $p_{(M, A)}$ from $p$ by changing the assignment of the macro $M$ to $A$. Let us define the marginal contribution of $M$ placed at $A$ by

$$c_p(M, A) = Cost(A) + \sum_{N \in E(M)} L(p_{(M, A)}(N)).$$

3. Use the method described in the solution of Problem 1 with cost function $c_p$ for $M, A$ to get a better assignment $p'$.

4. Continue with step 2 using the assignment $p'$ given in 3.

We run this procedure for a number of rounds.

**Remark 1.** With no further adjustment, the algorithm can easily result in an infinite loop as the following example shows:

Consider two macros and let the netlist be one single net connecting the two macros. During the run of the algorithm if the two macros are at different position, the macro with the higher cost would move to the area where the other macro is, because this decreases the total netlength and the total cost of the macros. The cost of this area increases until one of them moves to another place. As before the net connecting this macro to the other causes the other macro to move as well to the same area. Therefore the same process starts again. This shows that increasing the cost under the overlaps alone is not enough.

After each round in the algorithm if there are at least two macros at the same area, we increase the cost of that area. At the beginning of the algorithm we allow overlaps to get a better placement, basically allowing not only better, but slightly worse placements to prevent the algorithm to get stuck early in some local minimum. Later, we increasingly punish overlaps to prevent the loop in Remark 1. This is a kind of cooling process. If we set the increment rate properly, the macros will have the time to distribute evenly in the placement area, with small netlength. Later, this punishment goes to infinity, hereby enforcing a legal solution.
**Problem 3.** There is a set of macros $\mathcal{M}$, a netlist $\mathcal{N}$, a placement area $\mathcal{A}$ which is a discrete $n \times m$ grid and a netlength model $\mathcal{L}$ given. Each edge length of each macro is the multiple of the edge length of the grid. A placement $p : \mathcal{M} \to \mathcal{A}$ is legal if $S(M_1, p(M_1)) \cap S(M_2, p(M_2)) = \emptyset$, $M_1 \neq M_2$. During the run of the algorithm we allow non-legal placements. We only require that the final placement $p$ is legal. Our task is to find a legal placement which minimizes $\sum_{N \in \mathcal{N}} \mathcal{L}(p(N))$.

**Solution.** Here the places are not disjoint as in Problem 2, so during the run of the algorithm the macros can overlap partially as well. In this case we increase the cost under the intersection proportional to its size.

**Problem 4.** In the general setting the sizes can be real numbers.

**Solution.** To use the solution of Problem 3, we divide the placement area to a sufficiently fine discrete grid and use only natural numbers for approximation. We round up the edge length of each macro to the nearest multiple of the edge length of the grid.

### 4 Our global placer

The algorithm receives an initial placement (e.g. random with many overlaps) and then refines it to a global placement with minimized total netlength.

#### 4.1 The structure of the algorithm

Our algorithm consists of 4 steps as follows: At the beginning, we generate an initial placement, or we use the given one. Then, for a given number of rounds, we do the following.

1. We choose a macro $M$ randomly with original position $X_0$.
2. We generate $t$ possible new positions $X_1, \ldots, X_t$ around its original position, with $\text{move}_\text{macro}(M)$
3. We move the macro $M$ to the position $X_i$ that minimizes
   \[
   \text{weight}(M, X_i) + \text{NetLength}(E_{X_i}(M)) + \text{penalty}(M, X_i),
   \]
   where $E_{X_i}(M)$ is obtained by moving the macro $M$ to $X_i$.
4. We increase the weights under the overlaps of $M$ with every other macros.

#### 4.2 Notations

To discretize the problem, we consider the placement area to be a finite $n \times m$ grid. We can assume that $n = 2^p$, $m = 2^q$. These parameters are free to choose according to the available computing resources. Denote the size of the placement area by $A_x$ and $A_y$ (for the horizontal and vertical size). After we set $n, m$, we divide the placement area to an $n \times m$ grid. Our grid will consist of $nm$ squares of dimensions $x \times y = \frac{A_x}{n} \times \frac{A_y}{m}$. We record the cost as a stepfunction on the placement area which is constant on the cells of the grid. In other words we define the cost on the cells of the grid and not as a function on the placement area. Let $P_{i,j}$ be the weight under the $i$th square of the $j$th row. Denote by $\mathcal{M}_{n,m}$ the set of all $n \times m$ matrices. Define the inner product of two matrices (say $A, B \in \mathcal{M}_{n,m}$) as follows:

\[
A \ast B := \sum_{i=1}^{n} \sum_{j=1}^{m} a_{i,j} b_{i,j}.
\]

We represent a rectangle with its top-left and bottom-right corners. For a given rectangle $\tilde{R} = ((x_1, y_1), (x_2, y_2))$, $x_1 < x_2$, $y_1 < y_2$, we consider the slightly larger rectangle

\[
R = \left( \left[ \begin{array}{c} x_1 \\ x \end{array} \right] x, \left[ \begin{array}{c} y_1 \\ y \end{array} \right] y \right), \left( \left[ \begin{array}{c} x_2 \\ x \end{array} \right] x, \left[ \begin{array}{c} y_2 \\ y \end{array} \right] y \right) = \left( \left( a_1 x, b_1 x \right), \left( a_2 x, b_2 x \right) \right).
\]
This is the smallest rectangle of the grid covering \( \hat{R} \). From now on let every rectangle be given in the form:

\[
R = ((a_1x, b_1y), (a_2x, b_2y)) = ((a_1, b_1), (a_2, b_2)).
\]

The characteristic function of \( R \) is defined as the following \( n \times m \) matrix.

\[
A_R = (a_{i,j})_{i,j=1}^{n,m}, \quad a_{i,j} = \begin{cases} 1 & \text{if } a_1 < i \leq a_2 \text{ and } b_1 < j \leq b_2 \\ 0 & \text{otherwise} \end{cases}
\]

### 4.3 Data structure for the weights

We introduce a data structure by which we can calculate \([2]\) in \( O(\log(n) \log(m)) \) time, and also, we can increase the cost function by a constant under any rectangle \( R \) (as in Problem [3] in \( O(\log(n) \log(m)) \) time.

**Remark 2.** We have two operations on \( P \).

- \( f(R, P) = A_R \ast P \) returns the total cost under a given rectangle \( R \).
- \( g(R, P, w) \) increases each entry of \( P \) by \( w \) under the rectangle \( R \). (\( P := P + wA_R \))

In our algorithm, we use these operations in every round, therefore, we need to compute them fast.

We construct an orthogonal basis \( \{B_1, B_2, \ldots, B_k\} \) in this space \( (k = nm) \). For a given rectangle \( R \), in order to compute \( A_R \ast P \), we only need to know the products \( B_i \ast A_R \) for every \( i = 1, \ldots, k \). Increasing the entries under \( R \) by \( w \) in \( P = \sum_i \alpha_i B_i \) can be done by increasing the coefficients of the expansion \( \alpha_i = \alpha_i + wA_R \ast B_i \). We construct a base such that for every rectangle \( R \), there are only a few basis elements which are not orthogonal to \( R \), and hence the inner product can be computed in constant time. First, consider the one dimensional array \( P = (p_i)_{i=1}^{n} \) and let \( n = 2^p \). Define \( B_k^a \) as follows.

\[
B_k^a(j) = \begin{cases} 1 & (2k - 2)a < j \leq (2k - 1)2^a \\ -1 & (2k - 1)2^a < j \leq 2k2^a \\ 0 & \text{else} \end{cases}
\]

where \( a = 0, \ldots, p \), \( k = 1, \ldots, 2^{p-a-1} \), \( j = 1, \ldots, n \). We also consider the basis element \( B_1^0 = 1 \). For example, the elements for \( n = 8 \) are

\[
B_1^0 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad B_2^0 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad B_3^0 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad B_4^0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1 \end{bmatrix},
\]

\[
B_1^1 = \begin{bmatrix} 1 \\ 1 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad B_2^1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ -1 \\ -1 \end{bmatrix}, \quad B_3^1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \quad B_4^1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix},
\]

**Lemma 1.** Let \( s, t \in \mathbb{N}, s < t \). For a given \( R_{s,t} \), there are at most \( 2 \log(n) \) elements of the basis \( B_k^a \) for which \( R_{s,t} \ast B_k^a \neq 0 \).

**Proof:** It is easy to check that \( B_k^a \) is an orthogonal basis in \( \mathbb{R}^n \). Let \( R_{s,t} \in \mathbb{R}^n \) as in the Lemma:

\[
R_{s,t}(j) = \begin{cases} 1 & s < j \leq t \\ 0 & \text{else} \end{cases}
\]
If \( B_{k}^{a} \times R_{s,t} \neq 0 \) then either (3) or (4) or both holds:

\[
(2k - 2)2^a < s \leq 2k2^a
\]

(3)

\[
(2k - 2)2^a < t \leq 2k2^a
\]

(4)

For a given \( R_{s,t} \), the number of \( B_{k}^{a} \) for which (3), (4) or both holds, is at most \( 2\log(n) \). This completes the proof of the lemma.

It is easy to compute the scalar product of \( R_{s,t} \) and \( B_{k}^{a} \):

\[
Star_x(R_{s,t}, B_{k}^{a}) = -\min \{ s - (2k - 2)2^a, 2k2^a - s \},
\]

\[
Star_y(R_{s,t}, B_{k}^{a}) = \min \{ t - (2k - 2)2^a, 2k2^a - t \}.
\]

Then the scalar product of \( R_{s,t} \) and \( B_{k}^{a} \) is:

\[
R_{x,y} \times B_{k}^{a} = \begin{cases} 
\text{Star}_x(R_{s,t}, B_{k}^{a}) & \text{if only (3) holds} \\
\text{Star}_y(R_{s,t}, B_{k}^{a}) & \text{if only (4) holds} \\
\text{Star}_x(R_{s,t}, B_{k}^{a}) + \text{Star}_y(R_{s,t}, B_{k}^{a}) & \text{if (3) and (4) holds}
\end{cases}
\]

(5)

Now we can get a basis in \( \mathbb{R}^{n \times m} \) from the one dimensional case as follows. Let

\[
B_{k}^{a,b}(i, j) = B_{k}^{a}(i) \cdot B_{l}^{b}(j),
\]

where \( B_{k}^{a} \) corresponds to the basis in \( \mathbb{R}^{n} \) and \( B_{l}^{b} \) to the basis in \( \mathbb{R}^{m} \). It is not hard to check that \( \{ B_{k}^{a,b} \} \) is a basis in \( \mathcal{M}_{n,m} \). Following the argument of Lemma 41 for any rectangle \( R \), there are at most \( O(\log(n) \log(m)) \) basis elements \( (B_{k}^{a,b}) \) not orthogonal to \( A_{R} \). Furthermore, from (5), the scalar product \( A_{(x_1, x_2) \times (y_1, y_2)} \times B_{k}^{a,b} \) can be computed in constant time.

It is easy to see that \( B_{k}^{a,b} \) satisfies:

- \( \forall R \) rectangle, \( |\{ B_{k}^{a,b} : A_{R} \times B_{k}^{a,b} \neq 0 \}| = O(\log(n) \log(m)) \). Furthermore, we can find them in \( O(\log(n) \log(m)) \) time.

- \( \forall R \) rectangle \( A_{R} \times B_{k}^{a,b} \) can be computed in constant time.

### 4.4 Inflation

During the run of the algorithm, the cost of crowded areas may get too high, causing that all macros will avoid that area. Rather than waiting for the costs of all the other places to increase, we implement a cost reducer. It will reduce the differences between the high- and low-cost areas. We chose the method below because it can be easily implemented without further computation time. The best rate of inflation should be adjusted.

### 4.5 The increase(\( R, value \)) subroutine

Let \( \alpha = (\alpha_{k,l}^{a,b}) \) be a global variable denoting the coefficients of the basis elements \( B_{k,l}^{a,b} \) in the expansion of \( P = \sum_{k,l,a,b} \alpha_{k,l}^{a,b} B_{k,l}^{a,b} \). The increase(\( R, value \)) subroutine computes the scalar product of the basis elements \( B_{k,l}^{a,b} \) and \( A_{R} \), and increases the current coefficient of \( B_{k,l}^{a,b} \) with this product multiplied by \( value \). We repeat this for all \( B_{k,l}^{a,b} \):

**Algorithm 1** increase(\( R, value \))

```
for \{a = 0, \ldots, \log(n)\} do
  for \{b = 0, \ldots, \log(m)\} do
    for \{k, l : B_{k,l}^{a,b} \times A_{R} \neq 0\} do
      \( \alpha_{k,l}^{a,b} = \alpha_{k,l}^{a,b} + \text{scalar}(A_{R}, B_{k,l}^{a,b}) \times value \)
    end for
  end for
end for
```
In line 3, we can find the pairs \((k, l)\) in constant time as follows. For a given \(R\) there are at most 4 pairs \((k, l)\) such that \(\text{scalar}(A_R, B_{k,l}^{a,b}) \neq 0\). The possible pairs \((k, l)\) can be found easily from the coordinates of \(R\). Fix \(a, b\) and a rectangle \(R = ((x_1, y_1), (x_2, y_2))\), where \(x_1 < x_2, y_1 < y_2\). Let \(k_i, l_j\) be such that

\[(2k_1 - 2)^a < x_1 \leq 2k_1^2 2^a, \quad (2k_2 - 2)^a < x_2 \leq 2k_2^2 2^a, \quad (2l_1 - 2)^a < y_1 \leq 2l_1^2 2^a, \quad (2l_2 - 2)^a < y_2 \leq 2l_2^2 2^a\]

holds. The basis elements (with fixed \(a, b\)) possibly not orthogonal to \(A_R\) are \(B_{k_1,l_1}^{a,b}, B_{k_2,l_1}^{a,b}, B_{k_1,l_2}^{a,b}, B_{k_2,l_2}^{a,b}\).

### 4.6 The \(\text{cost}(R)\) subroutine

Here, \(\text{Round} \in \mathbb{N}\) is a global variable denoting the current round of the algorithm. The \(\text{cost}(R)\) function receives a rectangle \(R\) and returns the total cost of the cells inside this rectangle. This routine uses the basis expansion for the cost matrix \(P\) in order to compute the scalar product as follows:

**Algorithm 2** \(\text{cost}(R)\)

```plaintext
\text{cost} = 0;
\text{for } \{a = 0, \ldots, \log(n)\} \text{ do}
  \text{for } \{b = 0, \ldots, \log(m)\} \text{ do}
    \text{for } \{k, l : B_{k,l}^{a,b} \neq 0\} \text{ do}
      \text{cost} = \text{cost} + \alpha_{k,l}^{a,b} \cdot \text{scalar}(A_R, B_{k,l}^{a,b})
    \text{end for}
  \text{end for}
\text{end for}
\text{cost} = \text{cost} + \text{penalty}(\text{Round}, R);
\text{return cost;}
```

### 5 Heuristics

In this section, we discuss further parameters of the algorithm. We make suggestions for all parameters, but these should be experimentally adjusted.

#### 5.1 The \(\text{move}_\text{macro}(M)\) subroutine

This routine returns a new possible place for \(M\). As before, let \(A_x, A_y\) denote the horizontal and vertical size of the placement area \(\mathcal{A}\). The location of a macro \(M\) is given by its placement coordinates \((x, y)\). For a macro \(M \in \mathcal{M}\), let us denote the largest and smallest possible \(x\) coordinates for the macro \(M\) by

\[x_{\text{max}}(M) = A_x - \frac{\text{size}_x(M)}{2},\]
\[x_{\text{min}}(M) = \frac{\text{size}_x(M)}{2} - 1.\]

We define \(y_{\text{min}}(M), y_{\text{max}}(M)\) analogously. Let \(\gamma(x) = \exp(\log(x) \cdot U[0, 1])\) where \(U[0, 1]\) is a uniformly distributed random variable in \([0, 1]\). This distribution is our heuristic choice. The subroutine:

**Algorithm 3** \(\text{move}_\text{macro}(M)\)

```plaintext
a = \text{Rand}\{-1, 1\}
b = \text{Rand}\{-1, 1\}
x_{\text{new}} = \begin{cases} x - \gamma(x + 1) & \text{if } a = 1 \\
x + \gamma(x_{\text{max}} - x) & \text{if } a = -1 \\
y - \gamma(y + 1) & \text{if } b = 1 \\
y + \gamma(y_{\text{max}} - y) & \text{if } b = -1 \end{cases}
y_{\text{new}} = \begin{cases} x_{\text{new}} \end{cases}
\text{Return } x_{\text{new}}, y_{\text{new}}
```
5.2 The penalty(Step, R) function

Algorithm 4 penalty(Step, R)

\[
\text{cost} = 0
\]

for \(M \in \mathcal{M}, M \neq R\) do

\[
\text{cost} = \text{cost} + c \cdot \delta_{\text{Step}} \cdot \text{Circ}(M \cap R)
\]

end for

Return cost

Here, Circ(\(R\)) denotes the circumference of the rectangle \(R\), \(c\) is a constant and \(\delta_{\text{Step}}\) is a parameter.

5.3 The smooth_edge(E) function

We will consider the bounding-box model only. During the earlier stages of the optimization, when we compare different positions of a macro and we calculate the total distance of the wires, then we should consider that the positions of the neighbors are still rough. Therefore, it turns out to be useful to consider the positions of the neighboring pins with some uncertainty, namely, as distributions around their present positions. This can be expressed by using a smoothed version of the absolute value function of the difference in each coordinate. This tool was already used in the literature, it is common to approximate the bounding box model (1) with strictly convex functions which converges to the bounding-box netlength. One of them is the log-sum-exp function (see [2], [3], [4]):

\[
LSE_x(N) := \alpha \log \left( \sum_{p \in N} \exp \left( \frac{x(p)}{\alpha} \right) \right) + \alpha \log \left( \sum_{p \in N} \exp \left( -\frac{x(p)}{\alpha} \right) \right),
\]

and \(LSE(N) := LSE_x(N) + LSE_y(N)\). It is easy to see that \(LSE(N) \to BB(N)\), as \(\alpha \to 0\).

An alternative way is to approximate with \(L_p\) norms (see [5]):

\[
LP_x(N) := \sum_{p, q \in N} \left( (x(p) - x(q))^p + \alpha \right)^{1/p},
\]

and \(LP(N) := LP_x(N) + LP_y(N)\). \(LP(N) \to BB(N)\) holds again, if \(\frac{1}{\alpha} \to \infty\), \(p \to \infty\).

We used exponential functions, in a way similar to the log-sum-exp model, as follows:

\[
NL_x(N) = \frac{1}{\beta} \sum_{p \in N} \log \left( \exp \left( \beta x(p) \right) + \exp \left( -\beta x(p) \right) \right),
\]

and \(NL(N) = NL_x(N) + NL_y(N)\). It is clear that \(NL(N) \to BB(N)\) holds if \(\beta \to \infty\). We use

\[
\beta = \frac{\text{MaxRounds}}{\text{MaxRounds} - \text{Round} + 1},
\]

where \(\text{MaxRounds}\) is the number of rounds for which we want to run the algorithm. Formally the code of this subroutine is as follows:

Algorithm 5 smooth_edge(E)

\[
\begin{align*}
C_x &= \frac{1}{\beta} \log(\exp(\beta x(E)) + \exp(-\beta x(E))) \\
C_y &= \frac{1}{\beta} \log(\exp(\beta y(E)) + \exp(-\beta y(E)))
\end{align*}
\]

Return \(C_x + C_y\)

Notice that after many rounds, the edge length tends to the actual Bounding-box netlength.

5.4 Possible remaining overlaps

It is usually useful to stop the global placement before it removes all the overlaps. Our placer is ineffective in the very final stages of the algorithm, when the actual placement is almost legal, and only a few small overlaps should be eliminated. Therefore, we can get slightly better results if we stop the algorithm before the very final steps, and we use some other final legalization method, even a simple naive one. In our case, these final minor modifications were performed by hand.
6 Conclusions

In this paper we gave a heuristic algorithm for the NP-hard macro placement problem. The design of the algorithm is based on a primal-dual approach to a matching problem (see Section 3, Problem 1).

First, we implemented a special data structure to handle the dual (cost) function efficiently during the algorithm. This can record a multidimensional (in our case, 2-dimensional) discrete function, and performs efficiently the following two operations. It returns with the sum (integral) of the values in any rectangle, and it can increase the function with any constant in any rectangle. This data structure can also be useful for other purposes.

The second part includes the heuristics (see Section 5) inspired by the Hungarian Algorithm. We suggest an algorithm that iteratively revises the primal and the dual functions. Despite a pair of optimal primal-dual solutions do not exists, this causes problems only around the finalization of the placement. Our this heuristics seemed to perform well for finding good rough positions for the macros. Therefore, we used a natural continuous transition of the primal-dual method to a simple algorithm which just enforces disjointness. There were many minor details where we found nontrivial solutions which can be used in other problems, as well. All these together provide a flexible and robust algorithm for the VLSI placement problem, which can be easily optimized for different scenarios.

References

[1] B. Korte, D. Rautenbach, J. Vygen, BonnTools: Mathematical Innovation for Layout and Timing Closure of Systems on a Chip, Proceedings of the IEEE (2007), 95 (3), 555-572
[2] T.F. Chan, J. Cong, J.R. Shinnerl, K. Sze, M. Xie, mPL6: enhanced multilevel mixed-size placement, Proceedings of the International Symposium on Physical Design (2006), 212-214.
[3] J. Cong, G. Luo, Highly efficient gradient computation for desity-constrained analytical placement methods, Proceedings of the International Symposium on Physical Design (2008), 39-46.
[4] A.R. Agnihotri, P.H. Madden, Fast Analytic Placement using Minimum Cost Flow, Proceedings of the Asia and South Pacific Design Automation Conference (2007), 128-134.
[5] C.J. Alpert, T.F Chan, D.J. Huang, A.B Kahng, I.L. Markov, P. Mulet, K. Yan, Faster minimization of linear wirelength for global placement, Proceedings of the International Symposium on Physical Design (1997), 4-11.
[6] H. W. Kuhn, The Hungarian Method for the assignment problem, Naval Research Logistic Quarterly, 2 (1955) 83-97.