Faster and Better Global Placement by a New Transportation Algorithm

Ulrich Brenner
Research Institute for Discrete Mathematics
University of Bonn
Lennéstr. 2, 53111 Bonn, Germany
brenner@or.uni-bonn.de

Markus Struzyna
Research Institute for Discrete Mathematics
University of Bonn
Lennéstr. 2, 53111 Bonn, Germany
struzyna@or.uni-bonn.de

ABSTRACT

We present BONNPLACE, a new VLSI placement algorithm that combines the advantages of analytical and partitioning-based placers. Based on (non-disjoint) placements minimizing the total quadratic netlength, we partition the chip area into regions and assign the circuits to them (meeting capacity constraints) such that the placement is changed as little as possible. The core routine of our placer is a new algorithm for the TRANSPORTATION PROBLEM that allows to compute efficiently the circuit assignments to the regions. We test our algorithm on a set of industrial designs with up to 3.6 millions of movable objects and two sets of artificial benchmarks showing that it produces excellent results. In terms of wirelength, we can improve the results of leading-edge placement tools by about 5 %.

Categories and Subject Descriptors

B.7.2 [**Design Aids**]: Placement and Routing

General Terms

Algorithms, Design

Keywords

VLSI-Placement, Global Placement, Transportation Problem

1. INTRODUCTION

Placement is a crucial step in the physical design of VLSI chips. State-of-the-art VLSI chips consist of several millions of movable objects (circuits) that have to be placed disjointly in a given area. These circuits are connected by nets, and it is important for the complete design process to compute efficiently a placement that minimizes the total interconnect length and makes routing and timing optimization possible.

Often the placement task is divided into two parts: *global placement* where the circuits are spread over the chip area (without meeting the disjointness constraints exactly) and *legalization* (or *detailed placement*) where the circuits are moved to their final positions. Here, we consider global placement.

Many global placement algorithms apply a partitioning strategy. The main idea is to divide recursively the chip area into smaller parts and assign the circuits to them. Finally, when the regions are small enough, a legalization algorithm can be run. A widely used optimization goal for partitioning of the circuit sets is the minimization of the total cut size, i.e. the number of nets with pins in different parts (see, for example, [1], [6], [8], [9], [14]).

Other approaches ([17], [26]) combine partitioning-based placement with quadratic optimization. The algorithm presented in [26] computes a placement of the circuits that minimizes total quadratic netlength (ignoring disjointness) by solving a quadratic program (QP), and then asks for an assignment of the circuits to the subregions such that the total movement is minimized if we move each circuit to its region.

A placement approach without any partitioning steps is the so-called force directed placement ([13], [24]): Starting with the solution of a QP, repulsing forces between the circuits are computed and the QP formulation is modified according to these forces. This method is iterated until the overlaps are small enough.

Our Contribution

BONNPLACE combines quadratic optimization and top-down recursive partitioning. It applies ideas presented in [26] but contains a number of new contributions that improve both running time and quality of result:

- We describe a partitioning routine that can handle any number of subregions in each partitioning step. In addition, the subregions may have any arbitrary shape, and also the costs for moving a circuit to a subregion can be chosen arbitrarily. By introducing a new algorithm for the TRANSPORTATION PROBLEM, we will show how such a partitioning with minimum cost can be computed efficiently. So, the partitioning routine is much more flexible than the one described in [26] that can handle only four quadrands and has to use the L_1-distance to measure the movement of a circuit.
- We will show how the new TRANSPORTATION ALGORITHM can be used as a core routine for a partitioning based placement algorithm. Exploiting the flexibility of the approach, accurate models of the cost for moving a circuit to a region can be used. Even movements that would be necessary in further partitioning steps can be taken into consideration in advance.
- In addition, we will use the TRANSPORTATION ALGORITHM for local optimization steps that help improving the quality of result.
- We introduce a new hybrid net model which accelerates the QP computation significantly.
- We describe how time-consuming parts of our algorithm can be divided into subproblems that can be solved by parallel computation. This enables us to handle even the largest designs efficiently.

The remainder of the paper is organized as follows: Section 2 contains an overview of the entire algorithm. In Section 3, our new hybrid net model and an efficient implementation are explained. In Section 4, we describe the TRANSPORTATION PROBLEM and our new algorithm for unbalanced instances of this problem. Finally, Section 5 contains our experiments.

2. OVERALL ALGORITHM

We partition both the chip area and the set of circuits recursively by horizontal and vertical cut lines. Before each assignment of the circuits to subregions, we compute locations of the circuits in their regions such that quadratic netlength is minimized.

Let C denote the set of all circuits to be placed, and for a region r let $C(r)$ denote the subset of C assigned to region r. The general scheme of our placer is similar to other partition-based algorithms [17], [26]) and can be described as follows:

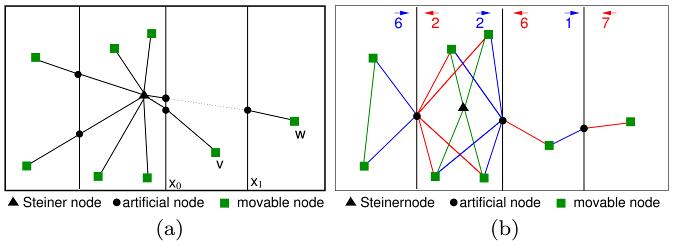
PARTITIONING-BASED PLACEMENT

```
① INITIALIZATION:
   window_set := {chip area}.
   $C$(chip area) := $C$.
② WHILE ( window size is big enough ) {
       Solve a QP to place circuits with minimum
       quadratic netlength inside their windows.
       Partition the windows into subwindows by
       adding cut lines
       FOR ( each window $r$ in window_set ) {
           MULTISECTION($r, C(r)$).
       }
       REPARTITIONING.
   }
③ LEGALIZATION.
```

```
MULTISECTION($r, C(r)$):
① Let $\{r_1, \dots, r_k\}$ be the set of subwindows of $r$.
② window_set := window_set$\setminus\{r\} \cup \{r_1, \dots, r_k\}$.
③ Apply the TRANSPORTATION ALGORITHM to partition
   $C(r)$ into $k$ subsets $C(r_1), \dots, C(r_k)$ meeting the
   capacity constraints of the regions and minimizing
   the total movement cost.
   Move circuits into the corresponding windows.
```

The core routine of our placer, the TRANSPORTATION ALGORITHM that computes the assignment of a set of circuits to a set of regions (meeting capacity constraints and minimizing movement cost) will be explained in Section 4.

Multi-terminal nets are represented by net models. Almost all quadratic placers (e.g. [26], [5]) replace each multi-terminal net by a clique or a star. It is easy to see that clique and star are equivalent by adjusting net-weights if there are no additional constraints. This fact has also been exploited in [26] and [23]. It is reasonable to use star instead of clique (at least for bigger nets) because using the star model leads to sparse matrices in the equation system, while clique may contribute $\frac{1}{2}|N|(|N|-1)$ non-zero entries for a net N. Additional problems occur, if we introduce linear constraints on circuit positions given by the partitioning. In a partitioning step, we assign each circuit to a certain rectangular window. We force each circuit to stay in its window in the QP solution by splitting the nets at the borders of the windows. In the computation of the x-coordinates, we split all nets at the vertical cut lines (while the horizontal cut lines are only considered in the computation of the y-coordinates). For example, in Figure 2 (a), the edge between pin w and the Steiner node is divided into two edges: one edge between the Steiner node and a fixed node at the coordinate x_0, and one edge between node w and a fixed node at the position x_1. This way, the QP solution will place each circuit between the boundaries of the window it is assigned to. However, if we split edges at cutlines, the star model and the clique model are not equivalent any more. Applying the star net model to such constrained problems can even be misleading as Figure 2 (a) shows: the position of v in a QP solution would not change if we removed w from the net. This inaccuracy of modeling leads to bad results when using the star net model instead of clique for a large number of nets, as experiments have shown. In Section 3, we will describe how to avoid such problems without loss of performance.



x0
x1
v
w
6
2
2
6
1
7
▲ Steiner node ● artificial node 🟩 movable node
▲ Steinernode ● artificial node 🟩 movable node
(a)
(b)

Figure 1: The star model and the new hybrid model

During global placement, the placement of the circuits is improved by a REPARTITIONING strategy that allows circuits to leave their windows. In a REPARTITIONING step, we consider a 2×2- or 3×3-window (i.e. sets of 4 or 9 regions that form a square), and compute new locations for the circuits in the window by minimizing quadratic netlength. Then, we run the partitioning method on the set of regions in the window (using the new locations). We replace the old placement in that area by this new placement if the netlength has improved. We run such a repartitioning step on each window and repeat the whole loop if it leads to a significant improvement. The repartitioning on 2×2-windows is faster while the repartitioning on 3×3-windows generally produces better results because of the slightly more global view. Experiments have shown that regarding larger windows than 3×3 increases drastically the running time but does not produce better improvements. Figure 2 shows an example for repartitioning on a 3×3-window. The circuits in this part of a chip are placed according to a QP solution, and their colors indicate the window they are assigned to; e.g., the darkest circuits are assigned to the window in the center.

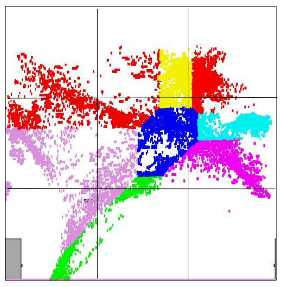


Figure 2: A multisection example on 9 regions

We call each iteration of the main loop of the algorithm a *level*. The main loop of our algorithm stops when the windows are "small enough". For row based designs, it is reasonable to stop the global placement loop when each window is a part of a circuit row that does not exceed a predefined length. Obviously, the number of levels depends on the number of horizontal and vertical cut lines that are inserted in each level. In our standard implementation we add one new cutline between two old cutlines in each of the early levels (so each window is partitioned into 2×2-subwindows). The TRANSPORTATION ALGORITHM would allow to add more horizontal or vertical cutlines in one level which could drastically reduce the number of levels, but then the running time for MULTISECTION would increase and, as experiments show, the result would get worse. However, in the last levels, when we have already a quite good spreading of the circuits, we can partition each window by two horizontal and one vertical cutlines without losing anything in terms of wirelength. For row-based designs, this helps reducing the number of levels.

To estimate the cost $d(c, r)$ for moving circuit c to region r, it is reasonable to take the distance between the position of c and the closest position in r circuit c could be placed at. Due to blockages in the chip area this distance can be significantly different from the distance between c and r. If many circuits are assigned to a region r, not all of them can be placed at the closest position in r. Therefore, we may have a look forward into the next level: We first partition each subwindow r into smaller regions that will occur in the next level and assign the circuits to these smaller regions. Then, we will assign a circuit to r if it was assigned to a subregion of r. This trick can improve the result especially in the first levels because we see in advance if a large movement inside a region r will be necessary in the next level.

After global placement, a legalization algorithm that removes the remaining overlaps has to be called. For our experiments, we apply the flow-based legalization approach described in [7] in order to obtain legal placements.

3. ACCELERATING THE ALGORITHM

The main part of BONNPLACE runtime is spent solving the QP and computing REPARTITIONING. We will describe in this section how these steps can be done efficiently.

3.1 Hybrid Net Model

We present a new net model, equivalent to clique, even in the presence of linear constraints, but leading to sparse matrices. The introduction of this new net model has a remarkable impact on runtime improvement and it is more memory-efficient than clique.

The hybrid net model consits of a vertical and a horizontal component, which can be computed independently. Thus, we restrict our description to the horizontal part, the vertical component works analogously.

Assume we are given a set $\mathcal{P}$ of intervals defined by the vertical cut lines. For each net N and each interval $P = [l_P, u_P] \in \mathcal{P}$ let N_P denote the set of pins in N whose x-coordinates have to be placed within P. Let $\lambda_{(P,N)}$ be the number of pins of N left to l_P and $\rho_{(P,N)}$ right hand to u_P. We define:

$$H_x(N, \mathcal{P}) := \frac{1}{|N|-1} \sum_{P \in \mathcal{P}} \Bigg(2 \cdot |N_P| \cdot \mathrm{STAR}(N_P) + \sum_{q \in N_P} \left[\lambda_{(P,N)} (x_q - l_P)^2 + \rho_P (x_q - u_{(P,N)})^2 \right] \Bigg),$$

where STAR(N_P) is the quadratic star net model applied to pins in N_P, and for a pin q, x_q is its x-coordinate. An example for the hybrid net model with the factors $\lambda_{(P,N)}, \rho_{(P,N)}$ for each (horizontal) part is shown in Figure 2 (b).

Using the above net model preserves all desirable matrix properties as positive definiteness, diagonal dominance and symmetry, but leads to same results as clique and maintains convexity [21]. Moreover, the QP matrix becomes sparse and can be decomposed along boundaries into smaller parts. Each of the resulting sub-QPs can be computed separately.

This decomposition of the matrices has an important impact on QP-computation runtime. We are able to shorten the running time of the entire placement algorithm by a factor of 2 due to this net model changes.

3.2 Parallelization

As each partial QP in an interval can be computed without any external information, we evidently can do it in parallel. A thread-pool of jobs performs the calculation and achieves remarkable speed-ups, which already on (8×8) windows exceeds 3.9 when using 4 CPUs and becomes even better in later QP.

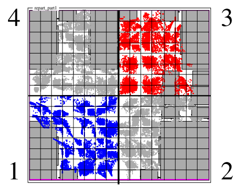


Figure 3: Parallelization of REPARTITIONING

For another part of the algorithm, the REPARTITIONING, we propose a geometric parallelization. Due to independent vertical and horizontal splits we cannot simply perform the parallelization on two disjoint windows for each pair. Hence, we divide the entire chip into quarters and perform the computation on the diagonally opposite parts simultaneously. For example, repartitionings in the upper right quadrant 3 in Figure 3 can be calculated in parallel with any window in the lower left area 1. Applying this strategy, we can gain about 40% of wall clock time for this part compared to simple parallelization of the QPs only.

4. TRANSPORTATION PROBLEMS

In this section, we will describe how the partitioning step that assigns circuits to subregions can be computed efficiently. For each circuit c and each region r we are given costs $d(c, r)$ for moving c to r, and we ask for an assignment of the circuits to the regions with minimum total costs such that no region contains more circuits than fit into it.

Even for two regions it is NP-complete to decide if a solution exists, since this problem contains the NP-complete PARTITIONING PROBLEM. Therefore, we relax the assignment problem by allowing to assign circuits fractionally to the regions. Let C be a set of circuits with sizes $size(c)$ for each $c \in C$ and R be a set of regions with capacities $cap(r)$ for each $r \in R$, and let $d(c, r)$ be the cost for moving circuit c to region r.

Then, we compute a fractional assignment of the circuits to the regions by solving the TRANSPORTATION PROBLEM. This can be formulated as a minimum-cost flow problem:

TRANSPORTATION PROBLEM

Instance:
- A directed graph G with vertex set $V(G) = C \dot\cup R \dot\cup \{s, t\}$ and edge set $E(G) = (C \times R) \cup (\{s\} \times C) \cup (R \times \{t\})$.
- Supply and demand values $b : V(G) \to \mathbb{R}$ with $b(s) = \sum_{c \in C} \mathrm{size}(c) = -b(t)$ and $b(c) = b(r) = 0$ for $c \in C$ and $r \in R$.
- Edge capacities $u : V(G) \to \mathbb{R}_+$ with $u((s, c)) := \mathrm{size}(c)$, $u((r, t)) := \mathrm{cap}(r)$, and $u((c, r)) = \infty$ for $c \in C$ and $r \in R$.
- Edge costs $u : V(G) \to \mathbb{R}_+$ with $\mathrm{cost}(s, c) = 0$, $\mathrm{cost}(r, t) = 0$ and $\mathrm{cost}((c, r)) = d(c, r)$ for $c \in C$ and $r \in R$.

Task Find a minimum-cost flow $f : E(G_i) \to \mathbb{R}_+$.

Of course, a positive flow $f((c,r)) > 0$ on edge (c,r) means that a fraction of size $f((c,r))$ of circuit c is assigned to region r. We may assume that $\sum_{c\in C} \text{size}(c) \leq \sum_{r\in R} \text{cap}(r)$ since otherwise no solution could exist. From now on, let $k := |R|$ and $n := |C|$.

For a solution f of the Transportation Problem, let S_f be the set of circuits which are not completely assigned to one region. Given an optimum solution f of the Transportation Problem and the set S_f we can easily converted f in time $O(k \cdot |S_f|)$ into an optimum solution f' such that $|S_{f'}| < k$ (see [25]). Such an "almost integral" solution is good enough for our aims because the remaining at most $k-1$ circuits can be assigned by any greedy strategy without changing the result too much.

Note that we have capacities only on edges incident to the artificial nodes s and t, so our instances can be regarded as uncapacitated. With the algorithm described in [19], uncapacitated Minimum Cost Flow Problems can be solved in time $O(|V(G)|(\log|V(G)|)(|E(G)| + |V(G)|\log(|V(G)|))) = O((n^2k + nk^2 + (n+k)^2\log(n+k))\log(n+k))$ which is $O(n^2\log^2 n)$ if k is fixed. However, for VLSI instances with several millions of circuits, the algorithm is much too slow, since its running time grows more than quadratically with n. There are other algorithms that exploit the special structure of the Transportation Problem. In [22] an algorithm is presented that solves the Transportation Problem in time $O(nk^2\log^2 n)$ which is $O(n\log^2 n)$ for constant k. So far, this was the best known algorithm for such unbalanced Transportation Problems. For constant k, we will improve this result by a factor of $\log n$, as our algorithm will solve the Transportation Problem in time $O(nk^2(\log n + k^2\log^2 k))$.

The idea of the algorithm is based on the well-known Successive Shortest Path Algorithm (see standard textbooks, e.g. [18], for an analysis and a proof of correctness). For our instances, the Successive Shortest Path Algorithm can be described as follows:

Successive Shortest Path Algorithm

Input: An instance (G,b,u,cost) of the Transportation Problem as constructed above.

Output: A minimum cost flow f in (G,b,u,cost).

```
①  $f(e) := 0$ for all $e \in E(G)$.
②  Let $C = \{c_1,\ldots,c_n\}$.
③  FOR($i = 1$ ; $i \leq n$ ; $i++$)
        WHILE($f((s,c_i)) < u((s,c_i))$)
            Find a shortest $c_i$-$t$-path $P$ in $G_f$.
            $\gamma := \min\{\min_{e\in E(P)} u_f(e), u((s,c_i))\}$.
            Augment $f$ along $P \cup (s,c_i)$ by $\gamma$.
```

Here, G_f is the residual graph of G for the fow f and $u_f(e)$ denotes the resisual capacity of an edge $e \in E(G_f)$ (the notation follows [18]).

We call an iteration of the main loop in step ③ a *phase* of the algorithm. In order to bound the running time of the algorithm, we have to bound the running time of a phase. However, even in the case of integer edge capacities, the number of augmentations for a single vertex $c \in C$ can be as big as $u((s,c))$ (if $\gamma = 1$ in each augmentation), so a single phase can have a running time that is exponential in the input size. Let $f_0(e) := 0$ for $e \in E(G)$, and let f_i be the flow at the end of phase i. In order to get a polynomial running time, we can replace a complete phase of the algorithm by computing a c_i-t flow of value $u((s,c_i))$ and minimum cost in the residual graph $G_{f_{i-1}}$. This method yields a polynomial running time, but $G_{f_{i-1}}$ will be as big as G. Fortunately, if we sort the nodes in C such that $u((s,c_1)) \geq u((s,c_2)) \geq \cdots \geq u((s,c_n))$, we do not have to consider the complete residual graph $G_{f_{i-1}}$ but a small subgraph G_i whose size does not depend on n. The vertex set $V(G_i)$ contains R, c_i, t and:

- For each vertex $c_{i'} \in C$ with $i' < i$, such that there is a vertex $r_j \in R$ with $0 < f_i((c_{i'},r_j)) < \text{size}(c_{i'})$: $c_{i'} \in V(G_i)$.
- For a vertex $r \in R$ let M_r^i be the set of all vertices $c \in C$ with $f_i((c,r)) = u((s,c))$. For each pair of vertices $r,r' \in R$ with $M_r^i \neq \emptyset$, $V(G_i)$ contains an arbitrary $c \in M_r^i$ with
$$\text{cost}((c,r')) - \text{cost}((c,r)) =$$
$$\min\{\text{cost}((c',r')) - \text{cost}((c',r)) : c' \in M_r^i\}.$$
(i.e. c is a cheapest element of M_r^i, if we have to move a circuit from r to r').

The edge set $E(G_i)$ contains the following edges:

- $((R\times\{t\}) \cup (\{t\}\times R)) \cap E(G_{f_{i-1}}) \subset E(G_i)$.
- $((\{c_i\}\times R) \cup (R\times\{c_i\})) \cap E(G_{f_{i-1}}) \subset E(G_i)$.
- For each $c_{i'} \in C$ with $i' < i$ such that there is a $r_j \in R$ with $0 < f_i((c_i',r_j)) < u((s,c_i))$: $((\{c_{i'}\}\times R) \cup (R\times\{c_{i'}\})) \cap E(G_{f_{i-1}}) \subset E(G_i)$.
- Let $r,r' \in R$ be two vertices with $M_r^i \neq \emptyset$, and let $c \in M_r^i \cap V(G_i)$ be the corresponding vertex in $V(G_i)$. Then $\{(c,r),(r,c),(c,r'),(r',c)\} \cap E(G_{f_{i-1}}) \subset E(G_i)$.

The size of G_i depends on the number of vertices $c_{i'} \in C$ for which there is a vertex $r \in R$ with $0 < f_i((c_{i'},w)) < u((s,c_{i'}))$. However, as mentioned above, if there are more than $k-1$ vertices of that type then it is easy to find a flow f_i' of the same cost with at most $k-1$ vertices of this type. After each phase of the algorithm we will call a subroutine Adjust(f_i) that makes sure that there are at most $k-1$ vertices of that type.

Therefore, we have $|V(G_i)| \leq k+1+1+(k-1)+k\cdot(k-1) = k^2+k+1$ and $|E(G_i)| \leq 2(k+k+(k-1)\cdot k+2k\cdot(k-1)) = 6k^2$. Note that the size of G_i does not depend on n but only on k.

Transportation Algorithm

Input: An instance (G,b,u,cost) of the Bipartite Minimum Cost Flow Problem

Output: A minimum cost flow f in (G,b,u,cost).

```
①  $f(e) := 0$ for all $e \in E(G)$.
②  Sort the set of nodes in $C$ such that $C = \{c_1,\ldots,c_n\}$
    with $u((s,c_1)) \geq u((s,c_2)) \geq \cdots \geq u((s,c_n))$.
③  FOR($i = 1$ ; $i \leq n$ ; $i++$)
        Construct $G_i$.
        Compute a minimum cost flow $g$ in
        $(G_i \setminus \{s\}, b', u_f|_{E(G_i)}, \text{cost}_f|_{E(G_i)})$ where
        $b'(c_i) = u((s,c_i))$, $b'(t) = -b'(c_i)$ and
        $b'(v) = 0$ for $v \in (C \cup R) \cap V(G_i)$.
        Augment $f$ by $g$.
        Set $f((s,c_i)) = u((s,c_i))$.
        Adjust$(f)$.
```

Chip	Circuits	Opt	Result	Gap	Time
Peko01	12 506	0.82	0.97	18.3 %	0:01
Peko02	19 342	1.27	1.48	16.7 %	0:02
Peko03	22 853	1.51	1.78	17.8 %	0:02
Peko04	27 220	1.76	2.05	16.5 %	0:02
Peko05	28 146	1.95	2.27	16.2 %	0:02
Peko06	32 332	2.07	2.43	17.5 %	0:02
Peko07	45 639	2.89	3.37	16.7 %	0:03
Peko08	51 023	3.15	3.71	17.5 %	0:04
Peko09	53 110	3.65	4.24	16.2 %	0:04
Peko10	68 685	4.75	5.50	15.7 %	0:07
Peko11	70 152	4.72	5.49	16.2 %	0:06
Peko12	70 439	5.02	5.84	16.3 %	0:06
Peko13	83 709	5.89	6.85	16.3 %	0:07
Peko14	147 088	9.03	10.56	16.9 %	0:10
Peko15	161 187	11.60	13.46	16.0 %	0:13
Peko16	182 980	12.50	14.63	17.0 %	0:17
Peko17	184 752	13.50	15.72	16.4 %	0:17
Peko18	210 341	13.20	15.39	16.6 %	0:18

Table 1: The results for PEKO test suite 3.

Chip	Circuits	Opt	Result	Gap	Time
Peko01	125 060	8.2	9.6	17.2 %	0:08
Peko02	193 420	12.7	14.9	17.0 %	0:19
Peko03	228 530	15.1	17.7	17.2 %	0:19
Peko04	272 200	17.6	20.6	17.1 %	0:32
Peko05	281 460	19.5	22.6	15.9 %	0:30
Peko06	323 320	20.7	24.1	16.6 %	0:34
Peko07	456 390	28.9	33.6	16.1 %	0:45
Peko08	510 230	31.5	37.0	17.4 %	0:56
Peko09	531 100	36.5	42.5	16.5 %	0:58
Peko10	686 850	47.5	55.6	17.1 %	1:21
Peko11	701 520	47.2	54.8	16.2 %	1:11
Peko12	704 390	50.2	58.8	17.1 %	1:24
Peko13	837 090	58.9	68.4	16.1 %	1:44
Peko14	1 470 880	90.3	105.4	16.7 %	3:14
Peko15	1 611 870	116.0	133.6	15.1 %	3:46
Peko16	1 829 800	125.0	146.6	17.3 %	4:50
Peko17	1 847 520	135.0	156.9	16.2 %	5:06
Peko18	2 103 410	132.0	153.5	16.3 %	5:03

Table 2: The results for PEKO test suite 4.

Chip	Circuits	Feng Shui 2.4		BonnPlace		
		BB	Time	BB	Diff	Time
IBM01	12 506	2.41	0:03	2.26	- 6.2 %	0:06
IBM02	19 342	5.34	0:05	4.93	- 7.7 %	0:10
IBM03	22 853	7.51	0:06	7.01	- 6.7 %	0:11
IBM04	27 220	7.96	0:07	8.23	+ 3.4 %	0:13
IBM05	28 146	10.10	0:08	10.02	- 0.8 %	0:13
IBM06	32 332	6.82	0:10	6.55	- 4.0 %	0:13
IBM07	45 639	11.71	0:13	10.41	-11.1 %	0:20
IBM08	51 023	13.60	0:16	12.68	- 6.8 %	0:30
IBM09	53 110	13.83	0:15	13.27	- 4.0 %	0:35
IBM10	68 685	37.48	0:22	32.92	-12.2 %	0:32
IBM11	70 152	19.96	0:21	19.15	- 4.1 %	0:37
IBM12	80 439	35.57	0:23	31.90	-10.3 %	0:48
IBM13	83 708	24.95	0:16	24.31	- 2.6 %	0:48
IBM14	147 088	38.48	0:52	37.82	- 1.7 %	1:00
IBM15	161 187	52.14	1:27	49.31	- 5.4 %	1:25
IBM16	182 980	61.33	1:16	57.88	- 5.6 %	1:50
IBM17	184 752	70.60	1:44	66.65	- 5.6 %	3:19
IBM18	210 341	45.05	1:54	45.74	+ 1.5 %	1:29
Average					- 5.1 %	

Table 3: The results for ISPD '02 test suite.

Theorem 1. *The* Transportation Algorithm *solves the* Transportation Problem *in time* $O(nk^2(\log n + k^2 \log^2 k))$.

Proof: (Sketch) **Correctness:** The algorithm replaces all augmenting steps of one phase of the Successive Shortest Path Algorithm by one min-cost flow computation. The only issue that remains to show for the correctness is that the augmentation steps of a phase in the Successive Shortest Path Algorithm can always be computed in G_i and therefore the min-cost flow computation in a phase of the Transportation Algorithm can also be restricted to G_i. However, by the construction of G_i and since we sorted the circuits in non-increasing order, it is easy to see that there is, during each phase i of the Successive Shortest Path Algorithm, always a shortest c_i-t-path in the residual graph that uses only edges in G_i.

Running time: Obviously, step ① can be done in time $O(nk)$ and step ② takes time $O(n \log n)$. The construction of G_i can be done in time $O(k^2 \log n)$ for each iteration if one stores each set M_w^i in $k-1$ heaps: For each pair $r, r' \in R$ with $r \neq r'$, we use a heap to store the elements v of M_r^i with $\text{key}_{r,r'}(v) = \text{cost}(c, r') - \text{cost}(c, r)$. Each flow can be computed in time $O\big(|E(G_i)| \cdot \log |E(G_i)| \cdot \big[|E(G_i)| + |V(G_i)| \cdot \log |V(G_i)|\big]\big) = O(k^4 \log^2 k)$, and the flow can be adjusted in time $O(k|V(G_i)|) = O(k^3)$. To update the heaps after a flow augmentation, there are $O(k^2)$ remove-operations (at most one per heap) necessary. The number of insert-operations after a flow augmentation is also $O(k^2)$: only the elements of $V(G_i)$ can be inserted to a heap, and for each heap that stores a set M_w^i, at most one element $c' \in V(G_i)$ for which there is a vertex $r' \in R \setminus M_r^i$ with $f_{i-1}((c', r')) = u((s, c'))$ can be inserted to M_r^i. Since there are at most $k-1$ elements $c' \in V(G_i)$ for which there is no $r' \in R$ with $f_{i-1}((c', r')) = u((s, c'))$ and each vertex can be added to at most $k-1$ heaps, we need at most $O(k^2)$ insert-operations. Since no heap contains more than n elements, each operation can be done in time $O(\log n)$. Therefore, the $k(k-1)$ heaps can be updated in time $O(|V(G_i)|k \log n) = O(k^2 \log n)$. □

The algorithm is efficient not only from a theoretical point of view but also in practice: A run on 3.6 millions circuits and 9 regions take less than 90 seconds (on an IBM P650 with 1.45 GHz).

5. EXPERIMENTAL RESULTS

We tested BonnPlace on two sets of artificial benchmarks and large industrial instances.

As artificial benchmarks we used the PEKO test suites, a set of placement instances that are generated in such a way that an optimum placement is known (see [11]). Since our placer needs (like any other analytic placer) at least one pre-placed circuit or IO-pin, we used for our experiments the test suites PEKO3 and PEKO4 which contain boundary IO-pins.

The results of our experiments are denoted in Table 1 and Table 2. As the instances are not too big and we want to know how close we can come to the optimum, we ran the program with 3×3-Repartitionings on the PEKO chips. The running times shown in the table are wall-clock times for complete runs (including legalization) on four processors. All experiments presented in this section were made on an IBM P650 with 1.45 GHz, and all running times are given in hours and minutes. The numbers in Table 1 and Table 2 show that on these instances our netlength differs from the optimum by about 17%. So far, the best published results on the testsuites PEKO3 and PEKO4 presented in [10] were approximately 20 % away from the optimum.

The second set of artificial test cases we used for our experiments are the ISPD '02 benchmarks (see [1], [3] and [15]). They were derived from the ISPD '98 benchmarks (see [4]) which were the basis for the PEKO benchmarks, too. Therefore each chip in the ISPD '02 benchmarks has the same number of circuits as the corresponding chip in the PEKO03 benchmarks.

Chip	Circuits	Nets	[26]		BonnPlace					
					1 processor 2×2 repart		4 processors 2×2 repart		4 processors 3×3 repart	
			BB	Time	BB	Time	BB	Time	BB	Time
Jens	72 496	73 273	6.92 m	0:10	6.77 m	0:07	6.76 m	0:03	6.53 m	0:09
Christian	289 509	299 692	166.44 m	1:01	166.14 m	0:30	166.04 m	0:16	156.45 m	0:33
James	412 505	426 689	108.34 m	1:32	109.57 m	0:48	109.80 m	0:21	100.88 m	0:51
Sven	825 737	836 549	253.07 m	3:14	254.23 m	1:59	252.85 m	0:53	246.16 m	1:35
Alex	983 173	1 040 431	207.98 m	4:26	201.53 m	2:47	200.99 m	1:09	197.59 m	2:09
Sandra	1 336 370	1 390 333	340.55 m	6:54	328.65 m	3:20	328.42 m	1:24	318.36 m	3:20
Reinhardt	1 513 864	1 560 123	366.59 m	6:07	360.65 m	3:12	360.96 m	1:28	355.23 m	2:57
Nadine	1 654 756	1 704 507	375.73 m	9:33	379.40 m	4:23	382.42 m	1:58	364.05 m	3:59
Hardy	2 057 814	2 076 540	353.50 m	9:22	365.71 m	4:33	363.24 m	2:03	341.05 m	4:23
Ulrich	2 602 006	2 663 760	505.06 m	14:47	504.38 m	7:31	506.77 m	3:26	490.20 m	6:26
Fermi	3 649 013	3 663 964	378.52 m	22:11	368.98 m	11:39	368.00 m	6:14	355.51 m	8:34

Table 4: The results for the IBM instances.

For the ISPD '02 instances, no optimal solution or nontrivial lower bound is known, but compared to the PEKO benchmarks, the ISPD '02 chips are much more realistic, since they contain macros and the connectivities reflect the net structure of real-world chips in contrast to the PEKO benchmarks where all nets are local nets in an optimum solution. Table 3 summarizes our results for the ISPD '02 benchmarks. For a comparison, we show the results of the placer "Feng Shui 2.4" as reported in [16]. We cite the bounding box netlength ("BB") and the running time on an 2.5 GHz Pentium 4 workstation ("Time"). We do not cite the results for the different Capo versions ([1], [2]) and mPG [12] which were reported in [16] be cause they were all outperformed by Feng Shui 2.4. For our algorithm, the table shows the corresponding numbers for a four-processor run with 3×3-Repartitioning. In addition, we report the difference to the Feng Shui results ("Diff"). The numbers demonstrate that we can improve the Feng Shui results on 16 of the 18 benchmarks. The average improvement (computed via the geometric mean of the ratios) is 5.1 %. So far, the Feng Shui-results were by far the best published placements on the ISPD '02 benchmarks.

In addition, we tested BonnPlace on a set of recent ASICs from IBM Microelectronics. We compared to the global placement approach described by [26]. Table 4 gives an overview on our experiments on the IBM chips. The instance sizes range from 72 000 to 3.6 millions. We ran our program with the standard parameter settings sequentially and on four processors in parallel. We also tested our algorithm with Repartitioning on 3×3-windows (only in the parallel version). After the global placements, we used the algorithm described in [7] for a legalization. We report for each run the netlength after the complete placement ("BB") and the wall clock running time ("Time") for global placement. The experiments prove that even the sequential version of our program is much faster than the method presented in [26] (with very similar netlengths). With the parallelized version, we can even place instances with 3.6 millions of movable objects in 6:14 hours. Moreover, applying the 3×3-partitioning, we can improve the netlength by 4.7 % compared to [26].

6. ACKNOWLEGMENT

We would like to thank Prof. Jens Vygen for helpful remarks and comments.

7. REFERENCES

[1] S.N. Adya, I.L. Markov: Consistent Placement of Macro-Blocks using floorplanning and standard-cell placement. ISPD (2002) 12–17.

[2] S.N. Adya, I.L. Markov, P.G. Villarubia: On whitespace in mixed-size placement and physical synthesis. ICCAD (2003), 311-318.

[3] S.N. Adya, I.L. Markov: Combinatorial techniques for mixed-size placement. to appear in: ACM Transactions on Design Automation of Electronic Systems (2004).

[4] C.J. Alpert: The ISPD98 circuit benchmark suite. ISPD (1998) 85–90.

[5] C.J. Alpert, T. Chan., D. J.-H. Huang, I.L. Markov, K. Yan: Quadratic placement Revisited, Design Automation Conference (1997) 752–757.

[6] C.J. Alpert, A.B. Kahng: Recent directions in netlist partitioning: a survey. Integration, the VLSI Journal 19 (1995), 1–81.

[7] U. Brenner, A. Pauli, J. Vygen: Almost optimum placement legalization by minimum cost flow and dynamic programming. ISPD (2004) 2–9.

[8] M.A. Breuer: Min-cut placement. Journal of Design, Automation and Fault-Tolerant Computing 1, 4 (1977) 343–382.

[9] A.E. Caldwell, A.B. Kahng, I.L. Markov: Can recursive bisection alone produce routable placements? DAC (2000) 477–482.

[10] T. Chan, J. Cong, K. Sze, K: Multilevel generalized force-directed method for circuit placement. ISPD (2005).

[11] C.C. Chang, J. Cong, M. Xie: Optimality and scalability of existing placement algorithms. ASP-DAC (2003), 621–627.

[12] C.C. Chang, J. Cong, X. Yuan: Multi-level placement for large-scale mixed-size ic designs. ASP-DAC (2003), 325–330.

[13] H. Eisenmann, F.M. Johannes: Generic global placement and floorplanning. DAC (1998), 269–274.

[14] D.J. Huang, A.B. Kahng: Partitioning-based standard cell global placement with an exact objectice. ISPD (1997), 18–25.

[15] ISPD02 benchmarks: http://vlsicad.eecs.umich.edu/BK/ISPD02bench/

[16] A. Khatkate, C. Li, A.R. Angihotri, M.C. Yildiz, S. Ono, C.-K. Koh, P. Madden.: Recursive bisection based mixed block placement, ISPD (2004), 84–89.

[17] J. Kleinhans, G. Sigl, F. Johannes, K. Antreich: GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization, IEEE Trans. on Computer-Aided Design 10 (3), 356–365 (1991).

[18] B. Korte, J. Vygen: Combinatorial Optimization: Theory and Algorithms. Springer, Berlin 2002, second edition 2002.

[19] J.B. Orlin: A faster strongly polynomial minimum cost flow algorithm. Operations Research 41 (1993), 338–350.

[20] PEKO benchmarks: http://ballade.cs.ucla.edu/ pubbench/placement

[21] M. Struzyna: Analytisches Placement im VLSI-Design, Diploma thesis, University of Bonn (2004) (in German)

[22] T. Tokuyama, J. Nakano: Efficient algorithms for the Hitchcock transportation problem. SIAM Journal on Computing 24 (1995), 563–578.

[23] N. Viswanathan, C. Chu: FastPlace: efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model, ISPD(2004), 26 – 33.

[24] K. Vorwerk, A. Kennings, A. Vannelli: Engineering Details of a Stable Force-Directed Placer, ICCAD (2004), 7–11.

[25] J. Vygen: Plazierung im VLSI-Design und ein zweidimensionales Zerlegungsproblem. Ph.D thesis. University of Bonn (1997) (in German).

[26] J. Vygen: Algorithms for large-scale flat placement. Design Automation Conference (1997) 746–751.