VLSI Legalization with Minimum Perturbation by Iterative Augmentation

Ulrich Brenner
Research Institute for Discrete Mathematics
University of Bonn
Email: brenner@or.uni-bonn.de

***Abstract*—We present a new approach to VLSI placement legalization. Based on a minimum-cost flow algorithm that iteratively augments flows along paths, our algorithm ensures that only augmentations are considered that can be realized exactly by cell movements. Hence, the method avoids realization problems which are inherent to previous flow-based legalization algorithms. As a result, it combines the global perspective of minimum-cost flow approaches with the efficiency of local search algorithms. The tool is mainly designed to minimize total and maximum cell movement but it is flexible enough to optimize the effect on timing or netlength, too. We compare our approach to legalization tools from industry and academia by experiments on dense recent real-world designs and public benchmarks. The results show that we are much faster and produce significantly better results in terms of average (linear and quadratic) and maximum movement than any other tool.**

I. Introduction

Placement legalization is a crucial step in the physical design of VLSI chips. It is applied both as a last step of placement algorithms and after local timing optimization steps that destroy legality of a placement, e.g., by gate-sizing and repeater insertion. In both applications, we can assume that all non-standard cells have already been fixed, so only standard cells that have to be aligned to rows remain movable. In addition, we can assume that the movable cells are spread over the chip area and can be legalized by local moves (though there can still be regions with an area utilization of nearly 100 %). The main goal is to find a legal placement as close to the given placement as possible, so we want to minimize the movement of the cells. Since legalization has to be run several times in a timing-driven design process, it must be performed very fast.

A. Previous approaches

Many legalization algorithms apply a greedy search and move cells one after the other to free locations or un-congested areas, e.g. TETRIS [9], (also applied in NTUPLACE3 [3] and in [14]), ABACUS [18], MONGREL [11], FASTPLACE [21], and DRAGON [20]. The greedy approaches are fast and flexible and work well on instances with low global placement density, but on dense designs they can produce large movements.

Other algorithms apply a global optimization to get a guideline in which directions cells have to be moved. Mainly minimum-cost flow formulations ([2], [4], [7]) or, similarly, force-directed approaches [17] are applied. The advantage of these algorithms is that they have a global view on the whole instance instead of considering only small parts, so they may have a better chance to avoid local optima. On the other hand, the solution of the global optimization does not lead directly to a legal placement and the final result may differ significantly from the initial global solution. For example, diffusion-based legalization improves the spreading of the cells but does not remove all overlaps, hence a final legalization is still necessary.

Flow-based algorithms first decide in which directions cells have to be moved from over-full areas to areas with free capacity. Afterwards, the flow has to be realized, i.e. cells must be chosen to be moved along the flow edges. Since it is often not possible to find cells of appropriate size, this will in general not directly yield a feasible assignment of the cells to the different areas. In the case that flow seeps away in small slots between blocked areas, the solution of the flow problem may even be completely misleading. Several heuristics are applied to cope with this problem, in particular by iterating the method with different parameters as long as there are over-full areas ([2], [7]). Another issue of flow-based algorithms is that one has to predefine in advance the cost for moving cells from one area to another. As long as we do not know which cells can be chosen, these edge costs are hard to estimate.

B. Our contribution

In the first part of our algorithm, we partition the chip area into small bins and assign the cells to the bins. In order to get rid of over-full bins, we balance the assignment by an algorithm that computes a flow between the bins. Different to previous flow-based approaches, our algorithm does not first compute a flow and then try to find cells of appropriate size to realize the flow. Instead, we modify a well-known minimum-cost flow algorithm (the SUCCESSIVE SHORTEST PATH ALGORITHM) such that whenever it touches an edge, we always take care of the set of cells that can be used to realize the flow along this edge. In this way, we make sure that only flow augmentations are chosen that can be realized exactly, and they are realized before the next augmentation.

Due to the modification, we lose the property of finding an optimum flow (with respect to some arbitrary predefined edge cost function), and such a flow is not our goal, but we still have a global view as we always consider the whole chip in each augmentation.

The new approach has a number of advantages compared to previous algorithms:

- Different to previous flow-based legalization methods, there is no need for arbitrary decisions concerning edge

978-3-9810801-8-6/DATE12/©2012 EDAA

costs and capacities. Both edge costs and capacities are computed on-line during the algorithm (and will be modified several times).
- We allow to assign cells fractionally to neighbouring bins of the same row, and hence we can work with small regions without using complicated data structures as proposed in [2].
- As we only apply augmentation steps that can be realized exactly, there are no rounding effects that could force the actual cell movement to deviate drastically from the flow solution.
- Whenever we decide to ship cells from one area to another, we always know which cells can be used for that shipment. Hence, our algorithm is very flexible in terms of the objective function provided that the effect of a single cell movement to the function can be estimated efficiently. E.g., we can optimize (quadratic, linear or maximum) movement, (weighted) netlength, or length of a timing-critical path.
- Moreover, we can handle several additional constraints. For example, we can set upper bounds on single netlengths or the movement of single cells.
- In practice, the approach finds efficiently legalizations with very small maximum and average linear and squared movement as the experimental results will show.

Our legalization algorithm is part of the placement tool BONNPLACE where it replaced an older flow-based approach (see [2] for an description of the old legalization and [19] for an description of the corresponding global placement tool).

The remainder of the paper is organized as follows: In Section II, we give a short overview of the legalization problem and its optimization goals. In Section III, we summarize known results on the minimum-cost flow problem. Section IV is the core part of the paper and contains the detailed description of our new approach. In Section V, we present experimental results, and Section VI contains concluding remarks.

II. OBJECTIVE FUNCTIONS AND CONSTRAINTS

We may assume that the given illegal placement is optimized (in particular in terms of timing and routability), so we want find a legal placement as close to the input placement as possible. Therefore, our goal is to minimize cell movement. One may consider the total linear movement of all cells (or equivalently the average linear movement), but this does not reflect the fact that a small number of large movements may be much worse, e.g., for the timing results, than moving many cells only a little bit. Hence, the maximum movement and the total squared movement (in which we sum up for all cells the square of the distance between its given position and its output position) may be more relevant objective functions.

Instead of just staying close to the input placement, one can also try to optimize the overall placement objective functions directly. There is no reason why cell movements that improve the worst timing-slack or at least reduce netlength should be rejected. Our approach is able to take care of such objective functions, however, it should be noted that our method is not designed for improving a legal placement as a postoptimization but for finding a legal placement.

Constraints that have to be reflected in legalization are upper bounds on the lengths of specific nets (in particular IO-nets) and predefined areas in which certain cells must be placed (so-called movebounds, see [19]). Our algorithm can avoid violating these constraints as we can simply forbid to move specified cells out of a given area.

III. MINIMUM-COST FLOW PROBLEMS

Minimum-cost flows are often used as guidelines in standard cell legalization algorithms, and also our algorithm is motivated by a minimum-cost flow algorithm. For an introduction to the minimum-cost flow problem and standard algorithms we refer to textbooks, e.g. [1]. One of the simplest minimum-cost flow algorithms is the SUCCESSIVE SHORTEST PATH ALGORITHM. It starts with a zero flow and then searches for shortest paths in the residual graph from a supply node v to a demand node w by Dijkstra's algorithm ([6]). Once such a v–w–path P is found, the flow is augmented along P and the supply value of v and the demand value of w are decreased. This is iterated until there are no supply nodes any more. It is well-known that this algorithm finds a minimum-cost flow. There are more efficient minimum-cost flow algorithms, in particular Orlin's algorithm [15], but since the SUCCESSIVE SHORTEST PATH ALGORITHM works by successive augmentation, it is appropriate for our application because we can modify it in such a way that only augmentations are made that can be realized exactly by cell movements.

IV. OUR APPROACH

A. Notation

Let a zone be a maximal part of a cell row that is either completely blocked or completely free. Our algorithm first subdivides each zone into bins. The core part of our algorithm consists of finding an assignment of the cells to the bins such that no bin contains more cells than fit into it. In our approach, cells are allowed to be fractionally assigned to free bins of the same zone. Hence, the objects that we move are fractional cells $\gamma = (c_\gamma, \rho_\gamma)$ which consist of a cell c_γ and a number $\rho_\gamma \in [0, 1]$.

For a cell, a zone or a bin x, let $\text{width}(x)$ denote its width. Given a (preliminary) assignment of the fractional cells to the bins and a bin v, let $\Gamma(v)$ be the set of fractional cells that are assigned to it. For a set Γ of fractional cells let $\text{width}_\rho(\Gamma)$ denote the total width of the fractions, so $\text{width}_\rho(\Gamma) = \sum_{\gamma\in\Gamma} \text{width}(c_\gamma)\rho_\gamma$. Let $\text{width}(\Gamma)$ denote the total width of the cells, so $\text{width}(\Gamma) := \sum_{\gamma\in\Gamma} \text{width}(c_\gamma)$.

B. Preprocessing

We start by partitioning all rows by a set of equidistant vertical cutlines (we assume w.l.o.g. that cell rows are horizontal). This yields also a partitioning of the zones into bins. Afterwards we unify neighbouring bins in the same zone if their total width is smaller than twice the distance between

two vertical cutlines. So, if there are no blockages, our bins induce a regular grid while, in general, the widths of the bins will vary. Now each cell is assigned completely to the bin that contains its global placement location. We call a bin that contains more cells than fit into it a supply bin and a bin with remaining free capacity a demand bin.

In order to reduce the total supply, we then run a greedy fractional reassignment in which parts of cells from a supply bin can be assigned to a neighbouring demand bin v in the same zone if their global placement location was close enough to v. This leads to a first fractional assignment but there will still be supply bins, and now our goal is to remove the supply. To this end, we connect neighbouring bins (in the same row or in neighbouring rows) by pairs of reverse edges and will ship cells along these edges.

C. Augmentation algorithm

Similar to the SUCCESSIVE SHORTEST PATH ALGORITHM we iteratively compute shortest paths from supply bins to demand bins by Dijkstra's algorithm [6]. Dijkstra's algorithm can be summarized as follows: Given a digraph G with edge weights $c : E(G) \to \mathbb{R}_{\geq 0}$ and two nodes $s, t \in V(G)$, it first initializes a set U by $U := V(G)$, and labels s with $l(s) := 0$ and all other nodes v with $l(v) := \infty$. Then, the algorithm iteratively chooses an element $v \in U$ with minimum label $l(v)$, removes v from U and sets $l(w) := \min\{l(w), l(v) + c((v, w))\}$ for all nodes $w \in U$ with $(v, w) \in E(G)$. This is iterated until t is removed from the set U.

At the end, for each node $v \in V(G) \setminus U$, the number $l(v)$ is the length of a shortest s-v-path (and, of course, one can easily keep track of the paths themselves).

In our variant of Dijkstra's algorithm we make use of the fact that we don't have to know the cost $c((v, w))$ of an edge (v, w) before v has been removed from U. Hence, $c((v, w))$ is only of interest when we have already found a shortest s–v–path. This enables us to make the cost of an edge (v, w) dependent on the set of cells that can be shipped from v to w. Therefore, we store on each edge $e = (v, w)$ that is used to reduce $l(w)$ to the smaller value $l(v)+c((v, w))$ a set $C(v, w)$ of candidate cells.

Algorithm 1 summarizes a single augmentation starting at a supply node s. The algorithm stores for each bin v a set $A(v)$ of fractional cells that can be used to be shipped to neighbouring bins. These have either been assigned to v before or are the candidates to be shipped into v. At the beginning, only $A(s)$ is defined. The number $b(v)$ denotes the supply (if $b(v) > 0$) or demand value (if $b(v) < 0$) of bin v. As we want to reduce only the supply of node s, we initialize all other b-values by at most 0 (line 2), so we see free capacities but we ignore supply values of bins different to s. However, if we send flow to a bin v, we increase its b-value accordingly (line 7), and if the b-value remains negative we can stop the path computation (line 10) because we have found an appropriate sink t. The mapping P which is set in line 20 is used to store the edges of the path along which the cells are shipped in lines 24 to 28.

Algorithm 1 Augmentation path algorithm

Input: A graph G whose nodes are the bins and whose edge set connects neighbouring bins. An assignment of the cells to the bins. A supply node s.
Output: A reassignment of the cells to the bins such that s is no supply node any more.

1: Set $A(s) := \Gamma(s)$ and $b(s) := \text{width}_\rho(\Gamma(s)) - \text{width}(s)$.
2: Set $b(v) := \min\{0, \text{width}_\rho(\Gamma(v)) - \text{width}(v)\}$ for $v \in V(G) \setminus s$.
3: Set $U = V(G)$.
4: **while** $U \neq \emptyset$ **do**
5: Let v by an element of U with minimum $l(v)$.
6: **if** $v \neq s$ **then**
7: $b(v) := b(v) + \text{width}_\rho(C(P(v), v))$.
8: $A(v) := \Gamma(v) \cup C(P(v), v)$.
9: **end if**
10: **if** $b(v) \leq 0$ **then**
11: Set $t := v$.
12: BREAK.
13: **end if**
14: Set $U := U \setminus \{v\}$.
15: **for** $w \in V(G)$ with $(v, w) \in E(G)$ **do**
16: Based on $A(v)$ choose a set $C(v, w)$ of fractional cells of size at least $b(v)$ with smallest movement cost to w.
17: Let $c(v, w)$ be the cost for moving the set $C(v, w)$ from v to w.
18: **if** $l(w) < l(v) + c(v, w)$ **then**
19: $l(w) := l(v) + c(v, w)$.
20: $P(w) := v$.
21: **end if**
22: **end for**
23: **end while**
24: Let $v_1, \ldots, v_k$ be a sequence of bins with $v_1 = s$, $v_k = t$ and $P(v_i) = v_{i-1}$ for $i = 2, \ldots, k$.
25: **for** $i = 1, \ldots, k - 1$ **do**
26: Move cells $C(v_i, v_{i+1})$ from v_i to v_{i+1}.
27: **end for**

In line 1, we initialize $b(s)$ by the total size of cells that have to be removed from s. Cells of this size will be shipped along a single path during the following augmentation. If $b(s)$ is large, it might be better to ship the supply of s in more than one iteration in order to distribute it to several demand bins. For this purpose, one can start with a number $b(s)$ that is smaller than $\text{width}_\rho(\Gamma(s)) - \text{width}(s)$. However, if the global placement tool did a good job, this will not be necessary, and in particular we made all experiments presented later without splitting the supply value of s.

A step that has to be described in detail is the choice of the set $C(v, w)$ of fractional cells to be shipped from v to w in line 16. If v and w are in the same zone, these are in fact fractional cells, so we can insert elements γ into $C(v, w)$ with $\rho_\gamma < 1$. Therefore, we may simply choose the cheapest elements from $A(v)$ (e.g. the leftmost cells if w is to the left of v) and reduce their ρ-value, if necessary. In particular, we can

always find sets of cells whose fractional size $\text{width}_\rho(C(v,w))$ is exactly $b(v)$.

On the other hand, if v and w belong to different zones, we can only ship cells completely because no cell can be distributed to bins of different zones. Hence, in that case $C(v,w)$ consists only of elements γ with $\rho_\gamma = 1$, so $\text{width}_\rho(C(v,w)) = \text{width}(C(v,w))$. Of course, $C(v,w)$ can only contain elements γ for which there is a $\gamma' \in A(v)$ with $c_\gamma = c_{\gamma'}$, but $\rho_{\gamma'} < \rho_\gamma$ is possible if $c_{\gamma'}$ has not been assigned completely to v. Then, shipping γ from v to w means to remove it from all bins (which must be in the same zone as v) and assign it completely to w. Note that in these cases we may have to choose cells such that $\text{width}_\rho(C(v,w)) > b(v)$. We compute the sets $C(v,w)$ by enumerating subsets of cells in $A(v)$.

The choice of the sets $C(v,w)$ is also the place where one may introduce other objective functions or constraints to cell positions. We know the recent positions for all other cells when we choose the sets $C(v,w)$, so we can, e.g., simply forbid to choose cells such that their movement would increase the length of a specific net over a critical value, or we can at least increase the cost for such choices.

D. Further steps

When we have found a fractional assignment of the cells to the bins, we have found as well an assignment to the zones. Hence, we can legalize all cells within their zones. For this purpose, we sort the cells in each zone according to their global placement locations and then apply the well-known clumping algorithm (see [8] and [13]) that finds a legalization with minimum total movement (with fixed ordering). Here, we minimize quadratic movement which enables us to run the clumping algorithm even in linear time. As a very final step, we allow cells to move to neighbouring bins in neighbouring rows if this decreases total quadratic movement. This step is performed in a greedy way by simply looking for some free space in the neighbouring bins.

E. Overall algorithm

Algorithm 2 gives an overview of the overall procedure. In the main loop (lines 5 to 9) we consider all supply bins in non-ascending order of their supply value since it seems to be reasonable first to get rid of the most congested bins.

Algorithm 2 Standard Cell Legalization

1: Subdivide chip area into bins.
2: Compute a fractional assignment of the cells to the bins.
3: Connect neighbouring bins by edges in a graph G
4: Insert all supply bins into a heap H (with the supply value as key).
5: **while** H is not empty **do**
6: Let s be an element of H with largest key.
7: Call AUGMENTATION PATH ALGORITHM on G and s.
8: Update keys in H
9: **end while**
10: **for** free zone z **do**
11: Legalize placement of cells assigned to z by the clumping algorithm.
12: **end for**
13: Allow cells to move to neighbouring bins.

V. EXPERIMENTAL RESULTS

We compared our legalization algorithm both to industrial and to academic tools. All experiments were performed on an Intel Xeon machine with 3.33 GHz with a Linux operating system.

In a first set of experiments, we ran tests on industrial designs from IBM microelectronics. Table I shows the chips that we used for these experiments. For each instance, we report the number of movable standard cells, the number of preplaced cells, which are fixed at their location, and the area utilization of the movable cell, i.e. the size of the movable cells divided by the total free area.

TABLE I
TESTBED 1: INDUSTRIAL ASICS

Chip	# Movable Cells	# Fixed Cells	Space Utilization
Chip1	323 311	80 267	56 %
Chip2	445 828	19 828	53 %
Chip3	565 620	54 208	51 %
Chip4	1 052 709	85	54 %
Chip5	3 942 782	41 533	49 %
Chip6	5 707 459	41 027	52 %
Chip7	5 780 500	83 699	53 %

On these chips, we compared us to two other approaches: the old legalization algorithm in BONNPLACE (see [2]) that is based on a minimum-cost flow formulation and contains a special postoptimization heuristic to reduce the largest movements, and the diffusion-based approach proposed in [17]. Both tools are used in industry.

For each chip, we produced two global placements with a maximum area utilization of 70 % and 95 %, respectively. The placements have been computed by the global placement part of BONNPLACE (see [19]) which observes density constraints very accurately.

The Tables II and III summarize our results.

In Table II we report the average quadratic (computed by the square of the Euclidean distances) movement and the running times. All running times reported in this paper are either in the format "mm:ss" or "h:mm:ss". Table III shows the maximum and average linear movement. In order to get technology-independent numbers, all movements are divided by the height of a cell row (or the square of the height of a cell row, for the squared movement).

The last two lines of the tables contain the average results for the other tools relative to our results (compared by the geometric mean of the ratios on the single instances), and the total running time.

On the sparse placements, the results for our tool are, in terms of average movement, only slightly better than the results of the diffusion-based tool, and both perform better than the flow-based method. However, for the maximal move-

TABLE II
ASIC TESTBED: QUADRATIC MOVEMENT AND RUNNING TIME

Chip	Density	Ours		Flow-based [2]		Diffusion-Based [17]	
		Average Quadratic	Time	Average Quadratic	Time	Average Quadratic	Time
Chip1	70	2.25	0:19	3.02	0:29	2.21	1:21
	95	4.79	0:38	5.41	1:26	6.57	1:26
Chip2	70	2.74	0:21	3.98	0:50	2.63	1:35
	95	5.64	1:03	6.87	2:06	6.48	2:07
Chip3	70	1.24	0:05	1.26	0:08	1.37	0:51
	95	2.20	0:07	2.29	0:14	2.81	0:59
Chip4	70	1.71	0:35	2.46	0:57	1.77	4:15
	95	2.90	0:43	3.74	1:16	195.06	14:45
Chip5	70	2.87	2:49	3.79	4:08	2.98	19:54
	95	5.31	4:01	6.51	9:41	4.39	18:05
Chip6	70	1.97	3:55	2.07	4:54	2.04	30:00
	95	3.92	4:54	4.14	5:35	5.45	34:51
Chip7	70	1.99	4:28	2.07	5:20	2.24	36:46
	95	4.10	3:52	4.34	5:27	5.90	45:12
Overall	70	1.00	12:32	1.22	16:46	1.04	1:34:42
	95	1.00	15:18	1.14	25:45	2.17	1:57:25

TABLE III
ASIC TESTBED: LINEAR MOVEMENT

Chip	Density	Ours		Flow-based [2]		Diffusion-Based [17]	
		Max	Avg	Max	Avg	Max	Avg
Chip1	70	102.13	1.52	112.24	1.72	102.13	1.52
	95	32.78	2.20	103.27	2.31	153.53	2.53
Chip2	70	35.91	1.69	56.08	1.96	80.76	1.65
	95	33.70	2.40	162.33	2.58	40.04	2.67
Chip3	70	11.25	1.17	9.64	1.19	7.75	1.24
	95	8.40	1.58	18.16	1.61	13.79	1.83
Chip4	70	9.50	1.35	10.54	1.54	8.77	1.36
	95	10.47	1.77	11.72	1.94	242.10	5.62
Chip5	70	42.68	1.66	64.01	1.86	60.48	1.66
	95	120.50	2.16	125.75	2.37	122.42	1.92
Chip6	70	30.36	1.45	32.36	1.49	27.49	1.49
	95	31.71	2.02	41.27	2.06	37.40	2.50
Chip7	70	63.38	1.46	55.64	1.48	93.42	1.53
	95	63.63	2.04	79.51	2.08	127.52	2.41
Overall	70	1.00	1.00	1.12	1.09	1.15	1.02
	95	1.00	1.00	1.80	1.05	2.43	1.30

ments, there is, even on the sparse instances, a larger quality gap between the result of our algorithm and the two other legalizers.

On average over the dense instances, the maximum displacement for the flow-based approach in [2] is by 80 % and for the diffusion-based approach [17] by 143 % larger than in our results. Even if we skip the dense version of Chip4, for which the diffusion-based approach produces extraordinarily bad results, the movement of the diffusion-based algorithm is larger by 22 % (average quadratic), 67 % (maximum linear), and 12 % (average linear), respectively. Moreover, the total running time of the flow-based legalizer is more than 30 % larger than the running of our algorithm, and the diffusion-based legalizer is by a factor of more than 7.5 slower than our legalizer.

In a second set of experiments, we compared our algorithm to academic legalization tools that are publicly available, namely the algorithm proposed by Ho and Liu in [10], NTUPLACE3 [3], FASTPLACE 3.0 [21], and XDP [5]. Unfortunately, we did not have access to the legalization tools TETRIS [9] and ABACUS [18], but the experiments published in [10] clearly show that they produced bigger movements than the new approach by Ho and Liu to which we compare ourselves. We also tried to compare to the history-based algorithm described in [4] but it did not run correctly as it ended with small overlaps (that could easily be removed), extremely large movements and running times of up to several hours on single instances.

For the comparison to the academic tools, we used the benchmarks chips from the ISPD 2006 placement contest [12] Table IV summarize the most important properties of these benchmarks. The maximal allowed area utilization in global placement was 90 %, and the global placement was computed again by the global placement algorithm in BONNPLACE.

TABLE IV
TESTBED 2: ISPD 2006 CHIPS

Chip	# Movable Cells	# Fixed Cells	Space Utilization
Newblue1	330 474	401	66 %
Newblue2	441 516	5 000	61 %
Newblue3	494 011	11 178	26 %
Newblue4	646 139	3 422	46 %
Newblue5	1 233 058	4 881	49 %
Newblue6	1 255 039	6 889	39 %
Newblue7	2 507 954	26 582	49 %

Tables V and VI summarize the results of the experiments on the ISPD benchmarks. As in the Tables II and III, we report average quadratic movement, running time, maximum displacement and average linear displacement. In the last row, we again show the average results (relative to our results) and the total running time, respectively.

For NTUPLACE and FASTPLACE we started two runs: in the first one, we disabled the detailed placement and stopped when the placement has been legalized (columns "Leg only"). In the second run, we enabled the detailed placement algorithms as postoptimization routine. Since these detailed placement parts of NTUPLACE and FASTPLACE try to optimize netlength while our goal is to minimize movement, we connected each cell by an artificial net to its location after global placement. All other nets were ignored. The results of these runs are contained in the columns "Leg + Detailed".

In XDP, legalization and netlength minimization are incorporated, hence we only have one run (with the artificial nets pulling cells to their initial locations).

For the legalization algorithm of Ho and Liu [10] that may consider netlength and movement simultaneously, we ignored all nets and only minimized movement.

On Newblue1, FASTPLACE ran into a segmentation fault both without and with detailed placement.

The results of the experiments on the ISPD 2006 benchmarks can be summarized as follows:

- Our algorithm is the fastest one and produces both in terms of linear, quadratic and maximum movement the best results.
- For average linear movement, the approach presented by

TABLE V
ISPD 2006 TESTBED: QUADRATIC MOVEMENT AND RUNNING TIME

Chip	Ours		Ho and Liu [10]		NTUPLACE [3]				FASTPLACE [21]				XDP [5]	
			Leg only		Leg only		Leg + Detailed		Leg only		Leg + Detailed		Leg + Detailed	
	Average Quadratic	Time	Average Quadratic	Time	Average Quadratic	Time	Average Quadratic	Time	Average Quadratic	Time	Average Quadratic	Time	Average Quadratic	Time
Newblue1	2.47	0:09	6.18	1:34	114.78	0:34	13.59	2:58	crashed		crashed		42.34	6:48
Newblue2	20.78	0:35	53.13	5:01	64.33	0:33	38.45	3:32	190.84	0:54	59.66	1:26	134.85	12:34
Newblue3	6.44	0:28	5.74	3:49	33.90	0:27	11.71	5:20	25.66	0:14	17.54	0:31	22.71	19:25
Newblue4	4.12	0:24	5.57	7:06	36.09	0:40	14.29	6:35	40.24	0:23	9.95	1:04	22.48	13:07
Newblue5	2.04	0:45	3.54	14:41	17.24	1:33	18.25	16:55	56.18	0:59	39.36	1:39	18.25	26:01
Newblue6	2.79	0:45	3.03	15:07	14.55	1:07	14.12	16:53	15.59	0:43	4.07	1:05	14.12	29:20
Newblue7	3.01	1:40	3.96	44:38	46.37	6:23	45.99	1:04:44	16.56	2:19	5.97	3:08	45.99	1:02:22
Overall	1.00	4:46	1.52	1:31:56	8.92	11:17	4.61	1:56:57	6.06	5:32	2.70	8:53	7.61	2:49:37

TABLE VI
ISPD 2006 TESTBED: LINEAR MOVEMENT

Chip	Ours		Ho and Liu [10]		NTUPLACE [3]				FASTPLACE [21]				XDP [5]	
			Leg only		Leg only		Leg + Detailed		Leg only		Leg + Detailed		Leg + Detailed	
	Max	Average	Max	Average	Max	Average	Max	Average	Max	Average	Max	Average	Max	Average
Newblue1	75.3	1.30	83.8	1.46	1176.8	8.52	121.4	1.92	crashed		crashed		267.0	1.96
Newblue2	66.8	3.77	429.8	3.99	2198.1	7.16	1716.5	4.24	683.3	8.67	553.3	4.04	285.6	5.76
Newblue3	21.3	2.11	58.8	1.95	246.9	3.68	162.8	2.38	233.8	3.87	155.1	2.62	221.7	2.64
Newblue4	72.5	1.70	90.0	1.72	88.6	5.28	99.6	2.12	760.3	3.69	255.0	1.71	100.2	2.35
Newblue5	44.6	1.30	154.8	1.34	154.1	2.50	154.5	1.81	788.2	2.76	752.0	1.41	154.5	1.81
Newblue6	46.6	1.49	64.3	1.49	74.4	2.71	70.8	1.90	500.9	3.00	536.3	1.51	70.8	1.90
Newblue7	233.6	1.28	207.8	1.32	291.0	2.52	293.9	1.84	564.8	2.68	422.1	1.32	293.9	1.84
Overall	1.00	1.00	1.94	1.02	2.43	2.40	3.12	1.29	9.03	2.08	6.49	1.07	2.83	1.39

Ho and Liu and FASTPLACE (with detailed placement) yield comparable results; on average over the seven chips their movement is 2 % or 7 % larger than ours, respectively.

- For quadratic and maximum movement the algorithm in [10] is second-best with a deviation of 52 % and 94 % respectively. All other tools produce even much larger quadratic and maximum movements.
- Compared to the second-best tool [10], our algorithm is faster by more than a factor of 19. The legalization part of FASTPLACE is almost as fast as our tool but even its average linear movement is more than twice of our movement.
- The versions of FASTPLACE and NTUPLACE with detailed placement produce better results than the pure legalization, even in terms of quadratic movement.

VI. CONCLUSIONS

We proposed a placement legalization algorithm that combines the global view of flow-based algorithms with the efficiency and flexibility of local search algorithms. Based on iterative augmentations, it produces in particular on dense instances legal placements with a very small perturbation and is significantly faster than previous approaches.

REFERENCES

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin: Network Flows: Theory, Algorithms, and Applications. Prentice Hall (1993).

[2] U. Brenner and J. Vygen: Legalizing a placement with mimimum total movement. TCAD (2004), 23, 12, 1597–1613.

[3] T.-C. Chen, Z.-W- Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang: NTUplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints. TCAD (2008), 27, 7, 1228–1240.

[4] M. Cho, H. Ren, H. Xiang, and R. Puri: History-based VLSI legalization using network flow. DAC (2010), 286–291.

[5] J. Cong and M. Xie: A robust mixed-size legalization and detailed placement algorithm. TCAD (2008), 27, 8, 1349–1362.

[6] E.W. Dijkstra: A note on two problems in connexion with graphs. Numerische Mathematik (1959), 1, 269–271.

[7] K. Doll, F.M. Johannes, and K.J. Antreich: Iterative placement improvement by network flow methods. TCAD (1994), 13, 10, 1189–1200.

[8] M.R. Garey, R.E. Tarjan, and G.T. Wilfong: One-processor scheduling with symmetric earliness and tardiness penalties. Mathematics of Operations Researchs, 13,2 (1988), 330–348.

[9] D. Hill: Method and system for high speed detailed placement of cells within an integrated circuit design. U.S. Patent, 6370673 (2002).

[10] T.-Y. Ho and S.-H. Liu: Fast legalization for standard cell placement with simultaneous wirelength and displacement minimization. VLSI-SoC (2010), 369–374.

[11] S.-W. Hur and J. Lillis: Mongrel: Hybrid techniques for standard cell placement. ICCAD (2000), 165–170.

[12] ISPD 2006 placement contest. `http://www.sigda.org/ispd2006/contest.html`

[13] A. Kahng, P. Tucker, and A. Zelikovsky: Optimization of linear placements for wirelength minimization with free sites. ASPDAC (1999), 241–244.

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

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

[16] M. Pan, N. Viswanathan, and C. Chu: An efficient and effective detailed placement algorithm. ICCAD (2005), 48–55.

[17] H. Ren, D.Z. Pan, C.J. Alpert, and P. Villarubia: Diffusion-based placement mitigation. DAC (2005), 515–520.

[18] P. Spindler, U. Schlichtmann, and F.M. Johannes: Abacus: Fast legalization of standard cell circuits with minimal movement. ISPD (2008), 47–53.

[19] M. Struzyna: Flow-based partitioning and position constraints in VLSI placement. DATE (2011), 607–612.

[20] T. Taghavi, X. Yang, B.-K. Choi, M. Wang, and M- Sarrafzadeh: Dragon2006: blockage-aware congestion-controlling mixed-size placer. ISPD (2006), 209–211.

[21] N. Viswanathan, M. Pan, and C. Chu: Fastplace 3.0: a fast multilevel quadratic placement algorithm with placement congestion control. ASPDAC (2007), 36–41.