On Legalization of Row-Based Placements

Andrew B. Kahng
CSE and ECE Departments
University of CA, San Diego
La Jolla, CA, 92093
abk@cs.ucsd.edu

Igor L. Markov
EECS Department
University of Michigan
Ann Arbor, MI, 48109
imarkov@eecs.umich.edu

Sherief Reda
CSE Department
University of CA, San Diego
La Jolla, CA, 92093
sreda@cs.ucsd.edu

ABSTRACT

Cell overlaps in the results of global placement are guaranteed to prevent successful routing. However, common techniques for fixing these problems may endanger routing in a different way — through increased wirelength and congestion. We evaluate several such techniques with routability of row-based placements in mind, and propose new ones that, in conjunction with our detail placer, improve overall routability and routed wirelength. Our generic two-phase approach for resolving illegal placements calls for (i) balancing the numbers of cells in rows, (ii) removing overlaps within rows through a generic dynamic programming procedure. Relevant objectives include minimum total perturbation, minimum wirelength increase and minimum maximum movement. Additionally, we trace cell overlaps in min-cut placement to vertical cuts and show that, if bisection cut directions are varied, overlaps anti-correlate with improved wirelength.

Empirical validation is performed using placers Capo and Cadence QPlace, followed by various legalizers and detail placers, with subsequent routing by Cadence WarpRoute. We use a number of IBMv2 benchmarks with routing information. Our legalizer reduces both Capo and QPlace placements' wirelength by up to 4% compared to results of Capo legalized by Cadence's QPlace in the ECO mode.

Categories and Subject Descriptors: B.7.2 [Design Aids]: Placement and routing
General Terms: Algorithms
Keywords: Detailed placement, legalization, min-cut placement

1. INTRODUCTION

With the advent of strong multi-level min-cut partitioners, recursive min-cut placement (e.g., [9, 22]) has emerged as a powerful and scalable technique that quickly produces placements with reasonable wirelength. Nevertheless, it often produces overlaps between several percent of cells. These overlaps must be resolved by detail placement, potentially increasing wirelength and impacting routability.

The origins of cell overlap in quadratic or analytical placements can be traced to relaxations of non-overlapping (or slot) constraints [14, 11, 21, 13]. Resulting illegal placements often have cells placed between rows, requiring a legalizer that can assign cells to rows [15]. On the other hand, global min-cut placers automatically assign cells to rows through partitioning, confining overlaps to same-row cells.

To formalize the legalization problem, we introduce the following notation. We assume a placement area composed of a set of r rows $\{\rho_1, \rho_2, \ldots, \rho_r\}$ and that for each row ρ_i, $\omega(\rho_i)$ is the total row width and $C_i = \{c_1, c_2, \ldots, c_m\}$[1] is the set of cells that are placed in ρ_i, where c_j is placed to the left of c_k if $j < k$. The horizontal cell position of cell c_j is given by $x(c_j)$, the vertical location by $y(c_j)$ and the cell width by $w(c_j)$. A row-based placement is called *legal* if and only if the placement meets the following two constraints:

1. No row ρ_i is assigned more cells than its capacity, i.e., $\sum_{j=1}^{|C_i|} w(c_j) \leq \omega(\rho_i)$.
2. No two distinct cells c_j and c_k in the same row overlap: $x(c_j) + w(c_j) \leq x(c_k)$ if $y(c_j) = y(c_k)$ and $x(c_j) \leq x(c_k)$.

We measure cell width in terms of the number of sites it occupies in a row. The *amount of overlap* in an illegal row-based placement is quantified by counting the number of consecutive cell pairs that overlap. Assuming that cells are sorted in their respective rows by their horizontal positions, this takes $O(n)$ time for n cells. On the other hand, calculating all pairs of cells that overlap takes $O(n^2)$. Several other metrics can be used to quantify overlaps: the number of cells involved in overlaps, the number of sites covered by at least two cells. Our preference is motivated by what is reported by MetaPlacer/Capo8.7 [9, 5], especially that this metric gives a good feel of how severe cell overlaps are in a given placement. We informally capture the overlap removal problem as follows:

Row-Based Placement Legalization Problem: Given a row-based cell placement, alter the cell positions to meet constraints (1) and (2) with minimum increase in wirelength or minimum total perturbation of cell positions.

[1] C_i should rather be written as $\{c_1^i, c_2^i, \ldots, c_m^i\}$ to avoid ambiguity in describing two different rows cells; nevertheless, we opt for the simpler notation in the text.

GLSVLSI'04, April 26 – 28, 2004, Boston, Massachussetts, USA.

In this work, we (1) examine the effect of cut-sequence choice on the amount of overlap, (2) propose site-granular solutions to solve the legalization problem, and (3) propose an accurate legalization procedure that optimizes a number of objectives while legalizing. These objectives include (min) Half-Perimeter Wirelength (HPWL), (min) total cell movement and (min) max cell location perturbation. The last two metrics are proposed as means to preserve the freespace distribution and hence routability of the design. We apply our legalizer to placements produced by QPlace5.2 from Cadence and Capo8.7 [9, 5] to legalize and/or improve wirelength by up to 4%.

The organization of this paper is as follows. Section 2 studies the effect of cut sequences on the amount of overlap and wirelength. In Section 3, we develop an overlap removal procedure and compare possible optimization objectives during legalization. We implement our overlap procedure and compare the various overlap objectives on the IBM benchmarks in Section 4. Finally, Section 5 summarizes the main contributions of this work and notes future directions for research.

2. NECESSITY OF OVERLAP REMOVAL FOR MIN-CUT PLACEMENTS

With any placement strategy, row-based global placements can potentially have overlaps. Though our present methods apply to legalization of *any* row placement, the min-cut context is of particular interest. In this section, we explain why overlaps are inherent in (wirelength-driven) min-cut placement. We examine the relationship between cut sequences, overlaps, freespace, and wirelength. We start by introducing some notation. Let the placement area be composed of r rows, and the total cell area (weight) is equal to w, with maximum individual cell weight of w_{max}. We also assume that all cells have single-row heights and that necessary condition for a placement with no overlaps - that the total capacity of the r rows is at least w - is satisfied.

Consider the basic case where there exists only $r = 2$ rows. For this case, the following fact is known ([18], page 268).

Fact 1. If the cells of total weight w are bisected into two halves to be placed in $r = 2$ rows, then a sufficient condition for a placement with no overlaps is that one row has area $\geq w/2 + w_{max}$. □

Fact 1 implies that in the worst case, one partition will contain cells with total weight $w/2+w_{max}$, while the other partition will contain cells with total weight $w/2 - w_{max}$. Since rows are arranged in a parallel fashion within the placement area, there is no way of simultaneously increasing the size of one row and decreasing the size of the other. This leads to the following lemma.

Lemma 1. For $r = 2$, a sufficient condition[2] a placement with no overlaps is to have freespace $\geq 2w_{max}$, with each row having $\geq w_{max}$ freespace. □

On the other hand, if a *single* row is partitioned vertically then no freespace is needed since the vertical cut line can be shifted to adjust for the partition weights, thus avoiding overlaps [1]. We can now generalize Lemma 1 to the case of $r > 2$ rows. Without loss of generality, we assume that $r = 2^k$ for some $k \in Z^+$.

[2]We assume a placer that will not introduce overlaps when it is not necessary to do so. Typically, placers will repartition blocks to best fit the partitioned cells [9, 1].

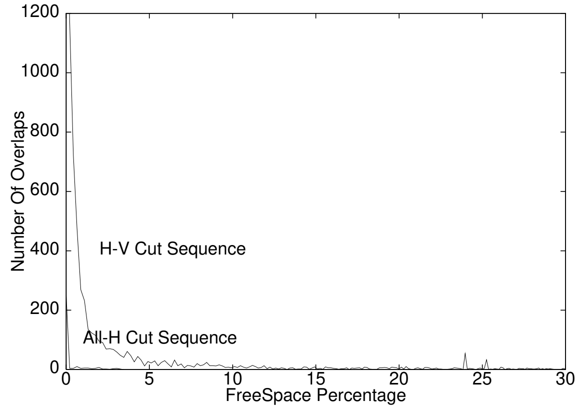

Figure 1: Relationship between available freespace and number of overlaps for the All-H and (alternating) H-V cut sequences.

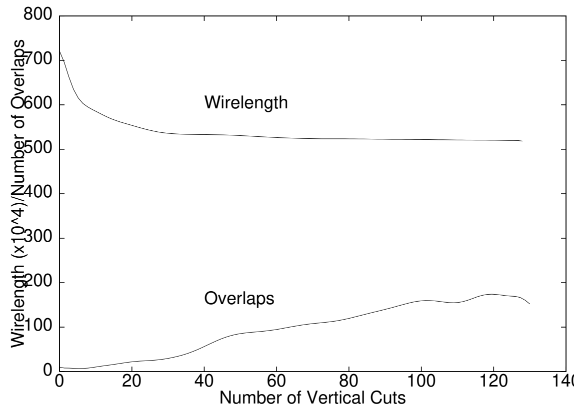

Figure 2: Relationship between number of vertical cuts, total wirelength and number of overlaps. Wirelength is reported in units x10^4.

Lemma 2. If cells of total weights w are recursively bisected to be placed in $r = 2^k$ rows then a sufficient condition for a placement with no overlaps is that each row has $2(1 - 1/r)w_{max}$ freespace, for a *total* freespace of $2(r - 1)w_{max}$ freespace in the layout.

Proof. The proof is by induction. For the base case $r = 2$, the lemma is true from Lemma 1. Assume that the lemma is true for some $r = n$; we show that it is also true for $r = 2n$. If the theorem is true for $r = n$ then we need $2(1 - 1/n)w_{max}$ freespace for each row. Hence, the total capacity of each row is $w/n + 2(1 - 1/n)w_{max}$. If each row is sliced into two, then from Lemma 1, each new row should have capacity $\frac{w/n+2(1-1/n)w_{max}}{2} + w_{max} = \frac{w}{2n} + 2(1 - \frac{1}{2n})w_{max}$, for a total of $2(r - 1)w_{max}$. □

Lemma 2 gives the amount of freespace sufficient to produce a placement with no overlaps if the cells are recursively bisected into $r > 2$ rows using only horizontal cuts. Note that vertical cuts will follow the horizontal cuts once individual rows are attained; however, in this case the vertical cuts produce no overlaps as mentioned above in the discussion of Figure 1. In practice, typical placers produce *alternating H-V* cut sequences. That is, horizontal cuts (H) are followed by vertical cuts (V), and vertical cuts are followed by horizontal cuts, and so forth. Some placers use the block aspect ratio in determining the cut direction with the resulting cut sequence typically alternating between horizontal and vertical cuts unless the initial placement area is far from being a square. The reason for such cut sequences is minimizing wirelength [19, 9, 24].

We expect that such alternating sequences produce larger number of overlaps since a vertical cut divides a set of n rows into $2n$ disconnected subrows. This is in contrast to a horizontal cut, which does not increase the number of subrows. An alternating H-V cut sequence stops making horizontal cuts after it has made r cuts, during which time r vertical cuts are executed. Hence, the total number of disconnected subrows is r^2. This larger number of disconnected subrows leads to harder packing resulting in increased overlap.

We empirically validate our arguments by examining the effect of vertical cuts, cut sequences and the freespace required for a placement with no overlaps. We use Capo [9, 5] as our min-cut placer. Given a benchmark, we gradually increase the amount of freespace and examine the effect of the introduced freespace on the amount of overlap. Figure 1 plots the overlap-freespace relationship for both the all-H and H-V cut sequences. Clearly, as available freespace increases, overlaps decrease. Furthermore, the all-horizontal cut sequence requires less freespace to produce non-overlapping placements than the H-V sequence, e.g., for ibm01 [4, 22] the all-H sequence produces non-overlapping placement at a threshold of approximately 3.45% freespace, while the H-V sequence requires around 22% freespace to eliminate overlap. From Lemma 2 and the benchmark characteristics, we calculate the freespace bounds. We find that for the all-H sequence, the calculated freespace bound using Lemma 2 is 3.7% which appears close to the actual empirical value. We also see that H-V cut sequences - in Capo8.7, the choice of the cut sequence depends on the block aspect ratio - produce a dramatically larger number of overlaps and that overlaps cease to exist after larger amount of freespace than the all-H cut sequence.

We also empirically study the relationship between cut sequences, overlaps and total wirelength. In this experiment, we control the number of vertical cuts executed during recursive bisection before single rows are attained. As more vertical cuts are allowed, we expect less total wirelength but more overlaps. We introduce into Capo a parameter that limits the number of vertical cuts taken. As we increase the threshold of vertical cuts that are allowed to be executed, we record both the total wirelength and number of overlaps from the placer's built-in reporting mechanisms. Figure 2 shows this relation for the ibm01 benchmark. The horizontal axis represents the allowable number of vertical cuts before individual rows are attained, while the vertical axis gives both the total wirelength (x10^4) and the number of cells overlapping. The curves show that the vertical cuts are a significant factor in producing overlaps yet are essential for wirelength minimization. We conclude:

Conclusion: The cut sequences that produce minimum overlap (all H- sequences) are exactly the cut sequences that produce largest wirelength, and vice-versa. Furthermore, vertical cuts on multiple rows (or subrows) are the reasons for a significant number of overlaps; nevertheless, they are essential for wirelength minimization.

Overlaps are thus a natural consequence of min-cut placement, and increasing freespace and allocating it in a uniform manner [8] helps reduce the amount of overlap. While it is attractive to produce legal placements from the start, our experiments (cf. Figure 1 and 2) indicate that this would require cut sequences far from optimal from the wirelength perspective. In addition, re-partitioning techniques [1] may reduce overlaps resulting from vertical cuts but typically never eliminate them especially for large benchmarks. With this conclusion in mind, it is natural for placers like Capo [9] or Dragon [22] to apply legalization during detailed placement (after global placement). In the next section, we address this problem while preserving important placement metrics such as wirelength and routability.

3. OVERLAP REMOVAL SOLUTIONS

We now develop an accurate overlap removal procedure and study various objectives for overlap removal. Our overlap solution is based on a two-phase approach. In the first phase, row capacities are met, and in the second phase, overlaps within all rows are removed.

3.1 Cell Juggling to Meet Row Capacity Constraints

In our first phase of overlap removal, we make sure that for each row, the sum of row cell sizes does not exceed the total row capacity. We develop a simple heuristic to achieve this objective with small impact on HPWL. We define *cell juggling* as moving cells *only* in the vertical direction. Since HPWL is independent for vertical and horizontal directions, cell juggling conserves the total horizontal wirelength.

Using the notation introduced in Section 1, we define the *row surplus* $S(\cdot)$ of some row ρ_i as the difference between the total row cell width $\sum_{j=1}^{|C_i|} w(c_j)$, and the row capacity $\omega(\rho_i)$, i.e., $S(\rho_i) = \sum_{j=1}^{|C_i|} w(c_j) - \omega(\rho_i)$. We measure cell width according to the number of sites occupied in a row. In the first step we calculate the surplus of all rows and rank the rows according to their surplus in a non-increasing order. Then for each row ρ_i with surplus $S(\rho_i) > 0$, we try to find a cell $c_j \in C_i$ such that juggling c_j to some row ρ_k with $S(\rho_k) < 0$ and $w(c_j) < |S(\rho_k)|$ introduces the minimum increase in the wirelength. We note that juggling is possible since we assume that the total cell area of the design is less than the total design area. We also note that for this phase we neglect any overlap resulting from juggling some cell c_j to row ρ_k since the second phase guarantees overlap removal within a row. The complete procedure for cell juggling is given in Figure 3.

3.2 Row legalization

In this subsection, we develop an accurate procedure that guarantees removal of cell overlaps within a row. We consider each row as set of ordered sites $S = \{s_1, s_2, \ldots, s_n\}$. For each row ρ_i, our goal is to place its set of cells C_i with no overlap, i.e., for any two cells c_j and c_{j+1}, $x(c_j) + w(c_j) \leq x(c_{j+1})$, where $x(.)$ indicates the leftmost site occupied by a cell. To produce a placement with no overlaps, we construct a directed acyclic graph $G = (V, E)$ as shown in Figure 4 with vertex set $V = \{0, \ldots, n\} \times \{0, \ldots, m\}$, and edge set

Input: An overlapped placement where row capacitances are not met.
Output: An overlapped placement where row capacitances are met.

1. **for** each $i = 1$ to r: calculate the surplus of row ρ_i as follows $S(\rho_i) = \sum_{j=1}^{|C_i|} w(c_j) - \omega(\rho_i)$.
2. Rank the rows in a non-increasing order according to the surplus.
3. **for** each row ρ_i where the $S(\rho_i) > 0$
4. **while** $S(\rho_i) > 0$
5. Initialize $\delta_{best} = \infty$.
6. **for** each cell $c_j \in C_i$:
7. **for** each $k = 1$ to r:
8. **if** $S(k) < 0$ and $|S(k)| > w(c_j)$ **then**
9. measure the increase in HPWL δ_{HPWL} if c_j is juggled to row ρ_k.
10. **if** $\delta_{HPWL} < \delta_{best}$ **then** $\delta_{best} = \delta_{HPWL}$, $\rho_{best} = \rho_k$, and $c_{best} = c_j$.
11. Move the cell c_{best} to row ρ_{best}, and update the surplus values of rows ρ_k and ρ_i.

Figure 3: The juggling procedure to meet row capacity constraints.

$E = \{(j, k-1) \to (j, k) \mid 0 \le j \le m, 1 \le k \le n\} \cup \{(j-1, k) \to (j, k + w(c_j)) \mid 0 \le j \le m, 1 \le k \le n - w(c_j)\}$

The set of edges E is composed of the union of horizontal edges and diagonal edges in G. We denote each edge by its start (tail) and end (head) point. A legal non-overlapping placement corresponds to finding a directed path from the origin vertex $(0, 0)$ to the final vertex (m, n). While any such path produces a legal placement, it is highly desirable to minimize the impact on placement metrics. We now introduce and compare three objectives.

1. **Minimum Perturbation (minPERB):** We seek to legalize a row using the minimum total cell displacements from the original locations. In this case, we label each diagonal edge by the difference between its start position $(j-1, k)$ in the graph and the actual cell position $x(c_j)$ in the row, i.e., the cost of edge $(j-1, k) \to (j, k + w(c_j))$ is $|x(c_j) - k|$.
2. **Minimum HPWL (minHPWL):** We seek to legalize a row using the minimum increase in HPWL. In this case, we label each diagonal edge starting at $(j-1, k)$ by the difference in HWPL from placing a cell c_j in position k rather than its current location of $x(c_j)$. We note that methods as [17, 6] also minimize HPWL using dynamic programming but not in the site-granular way that we do but rather using piece-wise linear properties of HPWL.
3. **Minimum Maximum Perturbation (minMAX):** We seek to legalize a row minimizing the maximum displacement from the original locations. We label the diagonal edges as in the minPERB case.

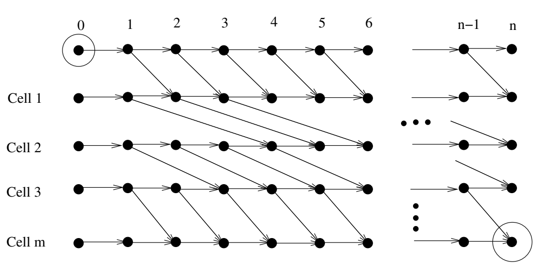

Figure 4: Shortest path computation for legalizing a row placement.

We label all horizontal edges by zero for all metrics. Realizing the objectives then corresponds to calculating the shortest path in G (except minMAX which corresponds to a path with the minimum max edge cost). Since G is directed-acyclic graph, the shortest path can be calculated using topological traversal of G in $O(mn)$ steps as given in Figure 5.

While applying one of the aforementioned objectives over all rows in a placement will certainly legalize the placement, we note that the **minHPWL** objective can be applied *iteratively* to further optimize the wirelength while retaining the legality of the placement. We refer to this iterative version of **minHPWL** by **minHPWLit**. Using this objective, legalization is repeatedly iterated over the whole chip until the improvement in wirelength drops below 0.1%. In general, we have found that iterating until there is no further improvement only enhances the solution quality by about 0.17% with respect to the solution quality of the stopping criterion of 0.1%. This improvement comes at the expense of additional CPU time, hence we opt to stop iterating if the improvement between the last two iterations is less than 0.1%.

As explained earlier, our overlap procedure is based on a detailed site by site consideration, nevertheless, we will see that runtimes are not of concern since legal placements are produced in practical runtimes. On the other hand, such detailed handling offers an advantage in optimizing metrics such as minMAX or minPERB. Also, it automatically solves a number of other practical issues such as handling of subrows in the placement area. These subrows may exist from placement of core components. These can be automatically handled by disallowing any diagonal edges in the sites of the rows that intersect with such core blocks. For example, if a core block occupies from sites s_1 to s_2 width and rows from some row ρ_f to row ρ_h then in legalizing any of these rows, we remove all the diagonal edges originating from any site k, where $s_1 \le k < s_2$ (These edges can be removed by simply setting their weight to ∞).

3.3 Experimental Results

We implement the proposed overlap removal strategy within Capo8.7 (the latest open-source version of Capo as of October, 18 2003). We use a binary built in the Meta-

Input: Row ρ_i, Set of cells $C_i = \{c_1, c_2, \ldots, c_m\}$ ordered from left to right.Placement objective.
Output: An non-overlapped placement of C_i attaining the input objective.

1. Initialize $cost(0,0) = 0$. **for** $j = 1$ to m: $cost(j,0) = \infty$. **for** $k = 1$ to n: $cost(0,k) = 0$
2. **for** $j = 1$ to m
3. **for** $k = 1$ to $n - w(c_j)$
4. **if** objective is minPERB, minHPWL **then** $cost(j,k) = \min(cost(j,k-1), cost(j-1,k-w(c_j)))$
5. **if** objective is minMAX **then** $cost(j,k) = \min(cost(j,k-1), max(cost(j-1,k-w(c_j))), |k - w(c_j) - x_{c_j}|)$
5. **return** $cost(m,n)$

Figure 5: Topological shortest path calculation for achieving various objectives.

Placer package of the UCLApack distribution [5], which is a wrapper around Capo that invokes a legalizer and a simple detail placer.[3] Capo's detail placer RowIroning is used in some experiments and turned off in others (-noRowIroning) — normally it optimizes a legal placement by selecting groups of 7-8 consecutive cells in each row and independently re-placing them using an optimal branch-and-bound procedure [7]. Such "optimization windows" are systematically moved through each row with a small step (2-3 cells). All reported HPWL values for Capo and QPlace are produced by the HPWL evaluator in from the GSRC bookshelf and closely agree with QPlace reports.

We run Capo on 2.4 GHz Pentium Xeon Linux workstation with 2 GB memory, while Cadence QPlace is run on Sun Ultra 10 machines with 1 GB memory (note that runtimes are not directly comparable). We empirically validate the results of previous sections with the following four experiments on a proprietary industrial circuit, as well as on "easy" versions of IBM version 2 benchmarks [23].

Experiment 1 applies the proposed overlap removal procedure to Capo's placements of IBMv2 benchmarks. For comparison, we legalize Capo placements using either Capo's legalizer or QPlace (version 5.2) in ECO mode (QP -ECO).

Results are reported in Table 1 for the easy instances. For space limitations, we do not tabulate results for hard instances. We observe that

- Capo's internal legalizer sometimes fails to completely remove overlaps, e.g., for ibm09 and ibm10 "hard" (not tabulated due to space limitations).
- minHPWLit gives the best improvement in HPWL but at the cost of runtime
- Both QPlace -ECO and our codes legalize all placements
- Our legalization and detail placement reduces HPWL by up to 4% compared to QPlace -ECO

Experiment 2 estimates the impact of legalization on routability by trying to route alternate placements of the ibm01 benchmark with Cadence's WRoute (version 2.3). Table 2 reports three relevant routability metrics. and suggests that wirelength minimization increases routing violations by up to 7.35% compared to the minPERB and minMAX objectives. Our experience indicates that such routing violations do not occur when the original placement was legal as we will shortly see in Experiment 4.

Experiment 3 compares our legalizers and detailed placers to existing tools: (i) QPlace in the -ECO mode, (ii) Capo's detail placer RowIroning [9] which optimally orders small groups of cells in rows. QPlace -ECO removes all overlaps, but our minHPWLit optimization further improves wirelength by up to 2.3%, according to results in Table 3. Routed wirelengths and the number of routing violations, measured after WRoute, indicate that minHPWLit does not impact routability. The additional use of RowIroning further improves HPWL by 2% (for a total of about 5%), but increases the number of routing violations so much that WRoute aborts. This is consistent with experiments in [9] which do not use RowIroning.

[3] The internal legalizer in MetaPlacer/Capo8.7 has been turned off by default due to a trivial coding bug, which we fixed for our experiments.

4. CONCLUSIONS

Global top-down row-based placers typically produce a small number of overlapping cells, that we trace to cuts perpendicular to rows. As naive legalization increases wirelength and can impact routability, we propose and evaluate several techniques for legalization and detailed placement. A particular issue that has not been addressed in the literature is that improving wirelength during detailed placement can be detrimental to routability.

Our contributions can be summarized as follows:

- We tabulate the influence of freespace and bisection cut sequence on wirelength improvement and the number of overlaps, showing that the latter two anti-correlate.
- We contribute a generic overlap remover based on a shortest-path computation.
- We empirically compare several possible legalization objectives, including their effect on routability.
- Our legalizer reduces the wirelength of Capo and QPlace placements by up to 4% compared to legalizing by QPlace in the -ECO mode.

5. REFERENCES

[1] S. Adya, I. Markov and P. Villarrubia, "On Whitespace and Stability in Mixed-Size Placement," in *IEEE Proc. of Intl. Conf. on Computer-Aided Design*, 2003, pp. 311-317.

[2] S. N. Adya et al., "Benchmarking for Large-Scale Placement and Beyond," *ACM/IEEE Intl. Symp. Phys. Design*, 2003, pp. 95-103.

[3] S. Akers, "On the Use of the Linear Assignment Algorithm in Module Placement," in *ACM/IEEE Proc. of Design Autom. Conf.*, 1981, pp. 13–144.

[4] C. J. Alpert, "The ISPD98 Circuit Benchmark Suite," in *ACM/IEEE Intl. Symp. on Physical Design*, 1998, pp. 18–25.

[5] UCLA Physical Design Tools, "http://vlsicad.cs.ucla.edu/software/PDtools,"

[6] U. Brenner and J. Vygen, "Faster Optimal Single-Row Placement with Fixed Ordering," in *Design, Autom. and Test in Europe*, 2000, pp. 117–122.

[7] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Optimal Partitioners and End-case Placers for Standard-cell Layout," *IEEE Trans. on Computer-Aided Des.*, vol. 19(11), pp. 1304-13, 2000.

[8] A. Caldwell, I. Markov and A. Kahng, "Hierarchical Whitespace Allocation in Top-down Placement," *IEEE Trans. on Computer-Aided Design* , vol. 22(11), 2003.

[9] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Can Recursive Bisection Alone Produce Routable Placements?" in *ACM/IEEE Proc. of Design Autom. Conf.*, 2000, pp. 477–482.

[10] T. Chan, J. Cong, T. Kong and J. Shinnerl, "Multilevel Optimization for Large-Scale Circuit Placement," in *IEEE Proc. of Intl. Conf. on Computer-Aided Design*, 2000, pp. 171–176.

[11] C. K. Cheng and E. S. Kuh, "Module Placement Based on Resistive Network Optimization," *IEEE Trans. on Computer-Aided Design*, vol. 4, July 1984, pp. 115–122.

[12] K. Doll, F. Johannes and K. Antreich, "Iterative Placement Improvement by Network Flow Methods," *IEEE Trans. on Computer-Aided Design*, vol. 13(10), 1994, pp. 1189–1200.

[13] H. Eisenmann and F. M. Johannes, "Generic Global Placement and Floorplanning," in *ACM/IEEE Proc. of Design Autom. Conf.*, 1998, pp. 269–274.

[14] K. M. Hall, "An r-Dimensional Quadratic Placement Algorithm," *Management Science*, vol. 17, pp. 219–229, 1970.

[15] D. Hill, "Method and System for High Speed Detailed Placement of Cells Within an Integrated Circuit Design," *US Patent 6370673*, 2001.

[16] S. W. Hur and J. Lillis, "Mongrel: Hybrid Techniques for Standard Cell Placement," in *IEEE Proc. of Intl. Conf. on Computer Aided Design*, 2000, pp. 165–170.

[17] A. B. Kahng, P. Tucker and A. Zelikovsky, "Optimization of Linear Placements for Wirelength Minimization with Free Sites," in *Asia and South Pacific Design Autom. Conf.*, 1999, pp. 241–244.

[18] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, 1st ed. John Wiley & Sons, 1990.

[19] K. Takahashi, K. Nakajima, M. Terai and K. Sato, "Min-Cut Placement With Global Objective Functions for Large Scale Sea-Of-Gates Arrays," in *IEEE Trans. on Computer-Aided Design*, vol. 14(4), 1995, pp. 434–446.

[20] J. Vygen, "Algorithms for Detailed Placement of Standard Cells," in *Design, Autom. and Test in Europe*, 1998, pp. 321–324.

[21] J. Vygen, "Algorithms for Large-Scale Flat Placement," in *ACM/IEEE Proc. of Design Autom. Conf.*, 1997, pp. 746–751.

[22] M. Wang, X. Yang and M. Sarrafzadeh, "DRAGON2000: Standard-Cell Placement Tool for Large Industry Circuits," in *IEEE Proc. of Intl. Conf. on Computer-Aided Design*, 2001, pp. 260–263.

[23] X. Yan, B. K. Choi and M. Sarrafzadeh, "Routabtility Driven White Space Allocation for Fixed-Die Standard-Cell Placement," in *ACM/IEEE Intl. Symp. on Physical Design*, 2002, pp. 42–47.

[24] M. C. Yildiz and P. H. Madden, "Improved Cut Sequences for Partitioning Based Placement," in *ACM/IEEE Proc. of Design Autom. Conf.*, 2001, pp. 776–779.

Circuit	Mode	Status	overlaps	HPWL	CPU (s)	Impr (%)
ibm01e	Capo raw	illegal	964	5.517	-	
	Capo leg	legal	0	5.586	-	
	QP -ECO	legal	0	**5.639**	1.0	
	minHPWL	legal	0	5.519	6.9	2.13%
	minPERB	legal	0	5.623	1.3	0.28%
	minMAX	legal	0	5.699	1.3	-1.06%
	minHPWLit	legal	0	**5.462**	39.1	**3.14%**
ibm02e	Capo raw	illegal	1502	1.599	-	
	Capo leg	legal	0	1.602	-	
	QP -ECO	legal	0	**1.624**	12.0	
	minHPWL	legal	0	1.579	15.2	2.77%
	minPERB	legal	0	1.604	2.1	1.23%
	minMAX	legal	0	1.607	2.2	1.05%
	minHPWLit	legal	0	**1.560**	76.3	**3.94%**
ibm07e	Capo raw	illegal	2816	3.706	-	
	Capo leg	legal	0	3.718	-	
	QP -ECO	legal	0	**3.756**	29.0	
	minHPWL	legal	0	3.695	37.5	1.62%
	minPERB	legal	0	3.730	8.5	0.69%
	minMAX	legal	0	3.760	8.1	-0.11%
	minHPWLit	legal	0	**3.674**	198.1	**2.18%**
ibm08e	Capo raw	illegal	3304	3.903	-	
	Capo leg	legal	0	3.912	-	
	QP -ECO	legal	0	**3.944**	34.0	
	minHPWL	legal	0	3.886	55.0	1.47%
	minPERB	legal	0	3.917	4.7	0.58%
	minMAX	legal	0	3.935	5.2	0.23%
	minHPWLit	legal	0	**3.862**	235.7	**2.08%**
ibm09e	Capo raw	illegal	3575	3.253	-	
	Capo leg	legal	0	3.275	-	
	QP -ECO	legal	0	**3.311**	34.0	
	minHPWL	legal	0	3.251	37.8	1.81%
	minPERB	legal	0	3.285	4.8	0.79%
	minMAX	legal	0	3.321	7.3	-0.30%
	minHPWLit	legal	0	**3.237**	139.3	**2.23%**
ibm10e	Capo raw	illegal	8811	6.222	-	
	Capo leg	legal	0	6.306	-	
	QP -ECO	legal	0	**6.349**	51.0	
	minHPWL	legal	0	6.246	324.2	1.62%
	minPERB	legal	0	6.312	286.5	0.58%
	minMAX	legal	0	6.434	291.5	-1.34%
	minHPWLit	legal	0	**6.226**	371.5	**1.94%**

Table 1: Results for IBM easy benchmarks. Capo raw is Capo's initial overlapping placement. Capo leg is Capo's legalizer. QP -ECO is Cadence QPlace legalizer. minHPWL is the proposed legalizer with min HPWL movement. minPERB, minMAX, and minHPWLit are the three objectives proposed in the text. CPU is the total legalizing time in seconds. HPWL values for ibm01 is $\times 10^7$, and HPWL values for ibm02-ibm10 is $\times 10^8$.

bench-mark	Objective	HPWL	Global Routing Metrics		Detailed Routing
			Overtrack	OverCapacity	violations
ibm01	minPERB	5.773	4489	3755	11743
	minMAX	5.846	4489	3755	11743
	minHPWLit	5.625	4616	3799	12602

Table 2: Effect of legalizing objective on routability of Capo's placements. For space limitations, we give results for only one benchmark; data from other benchmarks show similar trends.

benchmark	QPLACE		OUR		Impr
	wirelength	violations	wirelength	violations	(%)
ibm01	840769	0	820245	0	2.3%
ibm02	2098289	0	2113935	0	-0.56%
ibm07	4694388	0	4668722	0	0.63%
ibm08	5320330	0	5277750	0	0.93%
ibm09	3872250	0	3824083	0	1.29%
ibm10	7420895	0	7333763	0	1.21%

Table 3: Effect of legalizer on wirelength and routability of Cadence's QPlace placements on IBM easy instances.