Can Recursive Bisection Alone Produce Routable Placements? *

Andrew E. Caldwell, Andrew B. Kahng and Igor L. Markov
UCLA Computer Science Dept., Los Angeles, CA 90095-1596
{caldwell,abk,imarkov}@cs.ucla.edu

Abstract

This work focuses on congestion-driven placement of standard cells into rows in the *fixed-die* context. We summarize the state-of-the-art after two decades of research in recursive bisection placement and implement a new placer, called *Capo*, to empirically study the achievable limits of the approach. From among recently proposed improvements to recursive bisection, Capo incorporates a leading-edge multilevel min-cut partitioner [7], techniques for partitioning with small tolerance [8], optimal min-cut partitioners and end-case min-wirelength placers [5], previously unpublished partitioning tolerance computations, and block splitting heuristics. On the other hand, our "good enough" implementation does *not* use "overlapping" [17], multi-way partitioners [17, 20], analytical placement, or congestion estimation [24, 35]. In order to run on recent industrial placement instances, Capo must take into account fixed macros, power stripes and rows with different allowed cell orientations. Capo reads industry-standard LEF/DEF, as well as formats of the GSRC bookshelf for VLSI CAD algorithms [6], to enable comparisons on available placement instances in the fixed-die regime.

Capo clearly demonstrates that despite a potential mismatch of objectives, improved mincut bisection can still lead to improved placement wirelength and congestion. Our experiments on recent industrial benchmarks fail to give a *clear* answer to the question in the title of this paper. However, they validate a series of improvements to recursive bisection and point out a need for *transparent* congestion management techniques that do not worsen the wirelength of already routable placements. Our experimental flow, which validates fixed-die placement results by violation-free detailed auto-routability, provides a new norm for comparison of VLSI placement implementations.

1 Introduction

Standard-cell placement is a critical step in modern VLSI design automation flows and has captured the interest of both academic and industrial researchers for over two decades. Min-cut placement methods [3, 13] take a divide-and-conquer approach, recursively applying min-cut bisection to embed the netlist into the layout region. The *divide* step is now commonly implemented with the Fiduccia-Mattheyses heuristic [16, 4] or derivatives. The mincut placement tool CPlace [32] was heavily used in the design of IBM's RS 6000 and PowerPC chips (e.g., 601, 603, 604, 620 and derivatives) [33]. A breakthrough in min-cut partitioning came in 1997 when [19] and [2] validated the multilevel partitioning paradigm for hypergraphs with their highly successful implementations. Moreover, multilevel implementations have significantly improved since then, e.g., `hMetis` — the implementation presented in [19] — has improved by as much as 30% on some benchmarks in its more recent revisions. Yet, it is not known whether improved partitioning techniques still improve placement by recursive bisection, or whether they increase routability problems due to underutilization of routing resources along the cut lines.

Numerous other placement heuristics have been proposed as well, including *force-directed* and *quadratic placement* [37, 10, 31, 21] (PROUD, GORDIAN), *analytical placement with linear wirelength* [25] (GORDIAN-L), general non-linear optimization [26, 22] (RITUAL, POPINS), *flow methods* [12, 34, 18], *simulated annealing* [28] (TimberWolf) as well as rather recent exotic methods [15]. Most approaches have been extended to handle timing-driven placement in some way or another and, to make comparisons more difficult, have been combined with each other to form hybrids[1]. Among extensions to recursive bisection, applications of multi-way partitioning with geometric objectives [27, 17] and extensions for congestion-driven placement [24, 30] stand out ([24] is primarily directed at the variable-die context). Efforts to compare placements produced by different algorithms have been only marginally successful, due partly to the differences in *fixed-die* and *variable-die* formulations as well as non-trivial routability issues. A rare emprical comparison is described in [12].[2]

Successful attempts at "simpler but good enough" or "simpler but better" placement algorithms have been made more recently. In such works, one shows that certain techniques domi-

*Supported by a grant from Cadence Design Systems, Inc. Andrew E. Caldwell is now with Simplex Solutions, Sunnyvale, CA.

[1] E.g., [22] combines recursive min-cut bisection with non-linear timing models, [37, 31] combine partitioning and quadratic placement, and [34] uses network flow methods with quadratic placement.

[2] During the "TimberWolf Hunt", a combination of (analytical) GORDIAN-L [25] and DOMINO I (based on network flows) produced better solutions than TimberWolf (based on simulated annealing) using one-fifth the CPU time on the same variable-die benchmarks — both bounding-box and routed wirelength were measured in addition to runtime. On the largest design, with about 100,000 cells, the wirelength improvement was 22%. According to [12], GORDIAN-L alone produced better solutions than TimberWolf SC 5.4 and used only one-eighth the CPU time.

nate others, and/or builds a successful placer using only a small number of techniques (thus showing that other techniques are not necessary). For example, two papers presented back to back at the 1997 Design Automation Conference claimed that wirelength-driven VLSI placement can be successful (i) via recursive bisection and without analytical placers [1] and also (ii) via analytical placement and network flow methods without hypergraph partitioning [34]. Such conclusions are not necessarily contradictory. They help the practitioner to build simpler implementations and may point out obsolete approaches, just as recent works on ML-FM partitioning eclipsed the body of spectral partitioning.

While placement implementations are typically compared by their *half-perimeter wirelength*, we know of no published works that empirically confirm the correlation between better *half-perimeter wirelength* and better *routed wirelength* for *high-quality placements*. Of course, minimum wirelength with 100% auto-routability – *not* half-perimeter wirelength – is the objective of placement. Thus, our experimental flow includes a commercial placer and router from the same vendor and also allows using the router on placements produced by our Capo placer. If a placement can be routed without a single violation, we compare routed wirelength; otherwise, we consider *placement* a failure. We evaluate Capo on seven recent industrial circuits that can be placed and routed by the commercial tools we use. Capo succeeds on six of those and leads to better routed wirelength on five. In three of the six successful cases, better routed wirelength correlates with half-perimeter pre-routed wirelength. However, the only unroutable placement also has a better half-perimeter wirelength than the output of the commercial placer.

The main contributions of our work are the following:

- a strong recursive bisection implementation relying on a small number of recent portable techniques, but without any explicit congestion management;
- several new and easily implemented techniques for recursive bisection, including hierarchical tolerance computations and block splitting heuristics;
- empirical routability evaluation on recent industrial test cases with 0.1-30% whitespace, by running a leading commercial router on Capo placements;
- demonstration that improved mincut bisection still typically improves placements produced by recursive bisection;
- demonstration that *k*-way partitioning, "overlapping", analytical placement and explicit congestion evaluation are not necessary to produce routable placement of many circuits with 10-50K cells;[3] and
- demonstration of the need for "transparent" congestion management techniques that do not adversely affect routable placements.

2 Top-Down Placement

2.1 The framework

Top-down algorithms seek to decompose a given placement instance into smaller instances by subdividing the placement region, assigning modules to subregions, reformulating constraints, and cutting the netlist — such that good solutions to smaller instances (subproblems) combine into good solutions of the original problem. In practice, such a decomposition is accomplished by multilevel min-cut hypergraph partitioning [2, 19, 20] that attempts to minimize the number of nets incident to nodes in multiple partitions. Each hypergraph partitioning instance is induced from a rectangular region, or *block*, in the layout. Conceptually, a block corresponds to (i) a placement region with allowed locations, (ii) a collection of modules to be placed in this region, (iii) all nets incident to the contained modules, and (iv) locations of all modules beyond the given region that are adjacent to some modules in the region (considered as *terminals* with fixed locations). Cells inside the block are represented as hypergraph nodes, and hyperedges are induced by nets incident to cells in the blocks. Node weights represent cell areas, and partitioning solutions must have approximately equal total weight in all partitions. Otherwise, some blocks may have more cells than can possibly be placed inside without overlaps.

The top-down placement process can be viewed as a sequence of passes where each pass examines all blocks – blocks can be skipped, leaving them intact for the next level, or refines the block into two smaller blocks. These smaller blocks will collectively contain all the layout area and cells that the original instance did. When recursive bisection is applied, careful choice of vertical versus horizontal cut direction is important, as was shown in [30]: this choice may influence wirelength and routing congestion in resulting placement solutions. In this work we choose cut directions as in [5].

In attempting to improve basic recursive bisection, many researchers note that it eventually produces multi-way partitionings which could be alternatively achieved by direct methods using wirelength-like multi-way objectives [27, 17].[4] However, recent progress in efficient mincut partitioning has not been matched by wirelength-like optimization, and leading-edge recursive bisection typically produces very good solutions compared to other multi-way partitioning methods [20, 36].

2.2 Terminal propagation

Terminal propagation [13] is, perhaps, the most widely misunderstood concept in top-down placement, yet is essential to the success of the approach. When a particular placement block is split into multiple subregions, some of the cells inside may be tightly connected to external cells ("terminals") placed close to the subregions. Ignoring such connections allows a bigger discrepancy between good partitioning solutions and solutions that result in better placements. On the other hand, external terminals are irrelevant to the classic partitioning formulation as they cannot be freely assigned to partitions, due to their fixed status. A compromise is possible by using an extended formulation of "partitioning with fixed terminals", where the terminals are considered to be fixed in ("propagated to") one or more partitions, and assigned zero areas (original areas are ignored). Terminal propagation has been described in [27, 17, 5] and is typically driven by the geometric proximity of terminals to subregions/partitions. In this work we use terminal propagation as described in [5], which we have found difficult to improve.

2.3 Practical aspects and critiques

In a specific implementation of recursive bisection, the strength of the partitioning heuristics must be matched with the empirical difficulty of the partitioning instances — otherwise, either CPU

[3] We believe that the upper bound can be pushed significantly higher in the near future. Our results for circuits with smaller than 10K cells are also good, but we expect many techniques, including simulated annealing, to also be "good enough" for these. Similar qualifications of the "field of use" have been made for previous works (e.g., [26]).

[4] Most multi-way objectives degenerate to net-cut when there are only two partitions.

time or solution quality may be neglected. In this work we use flat FM instead of MLFM for instances with 200 or fewer cells.

Common critiques of top-down placement include underutilization of routing resources along higher-level cutlines, which can be seen as "bald areas" of top-down placements. It is possible, however, that (a) good placement solutions with "bald areas" exist, (b) bald areas are not harmful during routing, or (c) "bald areas" can be filled by detailed placement. Another criticism of recursive bisection suggests that it is not as relevant to placement as multi-way partitioning with wirelength objectives. However, in practice, recursive bisection often produces good solutions both with respect to multi-way netcut [20, 36] and wirelength objectives.

We also note that terminal propagation is only a crude approximation of the layout geometry — it is often impossible to say whether a particular cell will end up close to one subregion or another. In other words, top-down placement has to make a number of uninformed decisions which are irreversible because cells assigned to a block can never leave. As a possible workaround, [17] and other works use "overlapping", where neighboring blocks can be merged together and repartitioned. However, our experiments indicate that overlapping becomes *less useful* with stronger partitioners.

Yet another complication in top-down placement is the tight tolerance with which balanced hypergraph partitioning problems must be solved — a detailed description is available in [14], where a new technique is proposed to address the problem. However, their modifications to the Fiduccia-Mattheyses heuristic are rather complex and significantly increase run time in exchange for modest quality improvements. This work proposes a new approach, which is computationally trivial (has small constant-time overhead) and also uses easy-to-implement "repartitioning" techniques from [8].

3 Our Implementations and New Heuristics

We now describe our implementation of recursive bisection in Capo, which incorporates several recently proposed techniques as well as several novel elements. Besides such improvements, we greedily optimize cell orientations *after* recursive bisection. Such optimization reduces wirelength and considerably improves routability on some benchmarks.

3.1 Improved FM and MLFM implementations

We perform min-cut hypergraph partitioning for more than 200 cells using our modular implementation [7] of multilevel Fiduccia-Mattheyses heuristic [2, 19]. Its major component — an implementation [4] of the "flat" FM heuristic [16] — is also used independently for instances with 35-200 cells. Smaller instances are solved optimally with branch-and-bound.

In order to address the problem of small partitioning tolerance (described in Section 2.3), we utilize "uncorking" and "repartitioning" techniques recently proposed in [8]. The "corking" effect may degrade performance of an ordinary FM implementation when a large cell at the head of a bucket cannot be moved to the other partition without violating balance constraints. In this case, smaller cells further in the same bucket will not be considered for moves as the bucket is temporarily invalidated (see [4, 8] for details). "Uncorking" prevents large cells from being in the buckets in the first place and significantly improves expected solution quality without incurring runtime penalties. "Repartitioning" refers to chained FM calls on the same partitioning instance. The first call is performed with a much larger tolerance that requested — to ensure mobility of all cells. The tolerance gradually decreases to the original value in subsequent calls. Uncorking and repartitioning together significantly improve the flat Fiduccia-Mattheyses heuristic.

Our multilevel FM implementation [7] matches or improves upon the current versions of `hMetis`[19] in its trade-off between solution quality and runtime. Additionally, a single start of our implementation is extremely fast. While having several new elements, our implementation relies on a smaller set of heuristics than `hMetis`[19] and may be easier to replicate (for example, we always use exactly one V-cycle and no v-cycles; we use no "hyperedge removal"). Our experiments show that a strong multilevel partitioner is essential for achieving competitive placements of modern circuits; in particular, when we increased the number of MLFM starts for the first few placement levels in Capo, the average placement quality improved. In experiments below we vary the number of starts per partitioning. The default configuration of Capo uses the best of two partitioning sets, where a set is two independent starts followed by a V-cycle. An accelerated configuration uses only one such set per partitioning.

3.2 Block splitting heuristics

When comparing recursive bisection to direct multi-way partitioning, we note that it offers an easier control over the geometries of subregions. Indeed, different choices of vertical or horizontal cutlines and their precise location can result in a variety of placement block configurations. Choosing a complex block configuration for multi-way partitioning directly may be rather difficult. To exploit this advantage, we note that the choice of cutline location is significantly affected by its being across or along the rows. A straight-line cut perpendicular to rows can take a much larger set of locations, while straight-line cuts parallel to rows can effectively be only between rows. Therefore, when splitting a block by a vertical cutline, we first partition the hypergraph with a rather lax tolerance of 20% and *then* choose the cutline to equalize cell density in the resulting left and right subregions. Modifications to this "cutline adjustment" heuristic are needed to account for power stripes, fixed macro blocks and other obstacles in the layout (i.e., a prospective cutline that would cut too small a piece off some row will be vetoed by a specialized checker). Adjusting the cutline after partitioning effectively avoids partitioning with small tolerance, resulting in better partitionings and placements. Unfortunately, cutline adjustment is not possible when partitioning parallel to rows because the discreteness of the rows leaves few options for adjustment, and partitioning with small tolerances is inevitable in such cases. For more details and discussions of our block splitting techniques, see [5].

3.3 Hierarchical tolerance computations

We propose a new approach to address partitioning with small tolerances (a problem noted earlier). We find the maximal "reasonable" tolerance through careful accounting of the difference between *available site area* and *total cell area* in every placement block (*whitespace*). Our technical report [9] observes that *the requirement that cells do not overlap can be relaxed by requiring positive whitespace in every placement block*. It studies *relative whitespace* w, i.e., percentage of total cell sites that will not be occupied by cells. Relaxed constraints in terms of w become tractable very early in top-down placement, helping to ensure that actual non-overlapping constraints are satisfied. For example, the formulae below imply that when two placement blocks of identical geometry are partitioned with horizontal cutlines, the one having more whitespace can have a bigger partitioning tolerance.

Relative whitespace w in a block is proven to be a convex linear combination of relative whitespace values w_0 and w_1 in its sub-blocks [9]. Therefore, relative whitespace deteriorates in one of the sub-blocks and grows in the other. When w is very close to zero, one of the sub-blocks often ends up having *negative whitespace* ($\Rightarrow$ overlapping cells) due to number-partitioning reasons, especially when cell sizes vary greatly. Bounding *whitespace deterioration* by a fixed percent per level, [9] computes maximal whitespace deterioration for a given placement block using geometric summation. This leads to two formulae for partitioning tolerance τ in a given block:

$$\tau = \frac{(1-\alpha)w}{1-w}, \quad \alpha = \frac{\sqrt[n+1]{1-w}-(1-w)}{w\,\sqrt[n+1]{1-w}}, \quad n = \lceil \log_2 R \rceil \quad (1)$$

where w is *relative whitespace* in the current block (e.g., 0.1 for 90% utilization) and R is the number of rows in the block. Experiments in [9] suggest that such tolerance computation practically prevents cell overlaps in recursive bisection and can improve results of top-down placement. Finally, we note that the above considerations are not useful in the variable-die context as whitespace can often be added by enlarging placement blocks.[5]

3.4 Optimal partitioners and end-case placers

Our recent work [5] demonstrated the exceptional utility of optimal branch-and-bound partitioners on instances with up to 30-35 nodes. The Capo placer uses the same end-case placer for single-row blocks with up to 7-8 cells and also a somewhat improved optimal partitioner that splits blocks at several layers before end-cases. The thresholds for optimal partitioners and end-case placers are tuned so that those algorithms do not significantly affect the overall placement runtime (dominated by MLFM and flat FM). In particular, we introduce a time-out for branch-and-bound partitioners, upon which the best-seen but not necessarily optimal solution is given to flat FM as initial solution. Time-out is reached on partitioning instances that are pathological for branch-and-bound (other instances are solved optimally), and these appear typically very easy for FM.

Notably, the use of optimal partitioners addresses the problem of iterative partitioning with small tolerance (studied in [14] and in this work) *by avoiding the use of iterative partitioners*. This is especially convenient because, as explained in [5], the problem becomes more pronounced on small partitioning instances.

4 Empirical Validation

We first describe the differences between the fixed-die and variable-die placement methodologies and note that fixed-die placement is much more common today. This motivates our choice of the fixed-die placement context. We then describe our experimental flow and empirical results.

4.1 Fixed-die placement versus variable-die placement

Standard-cell row-based placement and routing can be performed in two major ways: variable-die or fixed-die. Variable-die methodology dates back to 2-layer metal (2-LM) processes and often minimizes chip area. Fixed-die methodology is the modern standard, and is appropriate to N-layer metal (N-LM) processes, $N > 3$. It is typically applied to design blocks rather than whole chips, therefore the block geometry and area are fixed — minimized are congestion and timing. Details of the two regimes are as follows.

Variable-die Inter-row spacings, row lengths and sometimes even the number of rows (e.g., in TimberWolf) may not be fixed *a priori*, but rather are determined during placement and routing. In particular, congestion is relieved by spreading rows (increasing the number of tracks between consecutive rows). Channel routers are typically used with the variable-die approach. Issues such as "feedthrough insertion" appear in relevant literature, because of the 2-LM standard-cell heritage.

Fixed-die The number/geometry of cell sites in cell rows are fixed before placement. In particular, all inter-row spacings are fixed. The fixed-die approach has its heritage from gate-array methodology. Area routers are typically used. (Routability thus becomes paramount: (i) congestion analysis and hot-spot removal, and (ii) floorplan (site map) optimization, become key parts of the P&R strategy.) The use of fixed-die approaches is driven by (hierarchical) design methodology: the presence of macros, a fixed floorplan, and fixed power and clock distribution networks together make the variable-die approach less relevant today. Fixed-die is also driven by the process: with N-LM processes blocks have high site utilization ($<$ 1% of whitespace is not uncommon); the use of "double-back" (shared power/ground rail) cell row architecture also fixes the row pitch.

LEF/DEF formats from Cadence Design Systems, Inc. are based on the fixed-die data model. Hence, all P&R tools that natively work with LEF/DEF follow the fixed-die data model.[6] Comparisons of placement results for fixed-die placers versus variable-die are not straightforward and should, at minimum, take the area and the shape of the die into the account.

The only set of sizable publicly available placement instances ("MCNC benchmarks" [23]) is variable-die, and certain row/layout information is missing. Unfortunately, the placement literature is full of conflicting interpretations of those benchmarks, which inhibits conclusive comparisons. Fixed numbers of rows used in [15] differ from those in [23], and there is no indication of the row lengths in final placements. TimberWolf [28] may adjust row spacing to improve routing, but other researchers report wirelength for rows packed tightly. Since the original testcases contain no pad locations, each pad is assigned a side of the layout region. Pad placement methodology (before, during or after placing core cells) can significantly affect final wirelength, but no recent works present their pad placement methods when reporting wirelength results. The combination of these effects can easily outweigh reported improvements due to new placement techniques, making MCNC testcases a questionable benchmarking platform until standard comparisons are established.

4.2 Our experimental flow

To evaluate routability, our experimental flow combines a fixed-die placer (which we run in congestion-driven mode) and a matching router from the same vendor. The flow allows swapping in our Capo placer for the commercial placer. While Capo always places cells in legal sites without overlaps with other cells, power stripes or other obstacles, more subtle violations

[5] Enlarged placement blocks can be used for minimizing congestion in the variable-die context (e.g., see [24]), but are harder to handle in the fixed-die context.

[6] One variable-die placer is distributed by InternetCad (formerly TimberWolf). While academic literature has used TimberWolf for comparison, to our knowledge all startup efforts as well as P&R offerings from major vendors (Avant!, Cadence, Mentor, Synopsys) are using the fixed-die model for standard-cell blocks.

(notably pin access collisions with M2 and higher geometries) can occasionally occur, i.e., Capo's legalization capability is incomplete. As a safety net, we run the industrial placer in its ECO mode to fix any violations in the Capo result. Such fixes are performed almost instantaneously, never improve wirelength and sometimes increase it by several percent. Currently, we do not have alternatives to this in our flow, but contend that such ECO "legalization" cannot possibly make Capo look better and does not affect the positive results of our experiments.

If a placement can be routed without a single violation, we look at the routed wirelength; otherwise, we consider *placement* a failure. We evaluate Capo on seven recent industrial circuits that can be placed and routed by the commercial tools we use. To empirically evaluate the routing difficulty of these designs, we also present the results of running our placer in "fast mode". Here Capo performs $1/2$ as many partitioning starts on each block, generally resulting in lower solution quality but faster runtime.

4.3 Experimental results

Statistics for seven industrial benchmarks in Cadence LEF/DEF format are given in Table 1 together with performance results of our placer in regular ("UCLA Capo") and fast ("Capo-Fast") modes. The performance of the industrial placer is given in the same table for comparison. Capo produces fully routable placements on six of these testcases, leads to better routed wirelength on four and ties on one (i.e., routed WLs within 1%). In three of the six successful cases, better routed wirelength correlates with half-perimeter pre-routed wirelength, however, the only unroutable placement has a better half-perimeter wirelength than that produced by the commercial placer. We observe that better routed wirelength often accompanies faster router runtimes and that more accurate partitioning is important to solution quality (the accelerated configuration does not achieve as good results as the regular one).

5 Conclusions and Open Questions

We evaluate the routability of placements produced by recursive bisection. Our implementation does not use explicit congestion management techniques, yet produces fully routable placements on six out of seven recent industrial benchmarks. In three of those six cases, better half-perimeter wirelength corresponds to better routed wirelength. The single unroutable placement solution does not fall into this trend as it has smaller half-perimeter wirelength than the routable placement produced by the commercial tool. This of course suggests that explicit congestion techniques may be useful on this testcase. However, such techniques should be implemented "transparently" so as to not adversely affect performance on other instances. Our experiments demonstrate that improvements in mincut partitioning are still conducive to better wirelength and congestion in top-down placement; however, this does not lessen the need for techniques that address congestion more directly.

Capo does not use analytical placement, and we make no judgments regarding the utility of analytic techniques for timing- or routability- driven placement. However, we believe that if a basic placement engine performs poorly on non-timing driven instances, congestion estimation and timing-related analytical techniques may be difficult to fine-tune as they have to minimize *total* wirelength. Besides, an improvement due to expensive congestion-driven backtracking or "overlapping" may be less valuable if it can also be achieved by careful ε-changes to "one-shot" recursive bisection.[7]

Our experiments on recent industrial benchmarks fail to give a clear answer to the question in the title of this paper. On the positive side, they validate a series of improvements to recursive bisection and point out a demand for transparent congestion management techniques that do not make routable placements worse. We also hope that our experimental flow, which validates placement results in the fixed-die regime by violation-free auto-routing, will provide a new norm for comparison of VLSI placement implementations.

Open questions for future research include

- Can a better wirelength- and congestion-driven placement approach be expected to lead to a better timing-driven approach (assuming congestion is still important)? In other words, is a timing-driven placer that performs poorly on timing-free instances likely to be eventually outperformed on timing-driven instances by placers that do well on timing-free?
- Are there techniques which can be seamlessly added to the recursive bisection approach so as to *prevent* (as opposed to correct) congested situations that cause unroutability?

Finally, another interesting and as yet unexplained effect seen in our results is the mismatch between weighted wirelength and routed wirelength objectives. While weighted wirelength predicts the value of routed wirelength reasonably well, Capo-Fast often produces smaller weighted wirelength, but larger routed wirelength. Therefore, the use of weighted wirelength as an optimization objective may be questionable.

References

[1] C. J. Alpert, T. Chan, D. J.-H. Huang, I. L. Markov and K. Yan, "Quadratic Placement Revisited", *DAC '97*, pp. 752-757.

[2] C. J. Alpert, J.-H. Huang and A. B. Kahng,"Multilevel Circuit Partitioning", *DAC '97*, pp. 530-533.

[3] M. A. Breuer, "Min-Cut Placement", *J. Design Autom. and Fault-Tolerant Comp.* 1(4) (1977), pp. 343-362.

[4] A. E. Caldwell, A. B. Kahng and I. L. Markov,

- "Design and Implementation of the Fiduccia-Mattheyses Heuristic for VLSI Netlist Partitioning", *Lecture Notes in Computer Science*, vol. 1619, Springer, 1999.
- "Design and Implementation of Move-Based Heuristics for VLSI Hypergraph Partitioning", in *ACM Journal of Experimental Algorithms*, 2000.

[5] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Optimal Partitioners and End-case Placers for Top-down Placement" *ISPD '99*, pp. 90-96.

[6] A. E. Caldwell, A. B. Kahng and I. L. Markov, "VLSI CAD bookshelf", 1999, `http://vlsicad.cs.ucla.edu/GSRC/bookshelf/`

[7] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Improved Algorithms for Hypergraph Bisection", *ASP-DAC 2000*, pp. 661-666.

[8] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Iterative Partitioning With Varying Node Weights", to appear in *VLSI Design*, 2000.

[9] A. E. Caldwell and I. L. Markov, "Hierarchical Whitespace Allocation", *Technical Report TR-200002*, UCLA Computer Science Dept., 2000.

[10] C. K. Cheng and E. S. Kuh. "Module Placement Based on Resistive Network Optimization", *IEEE Trans. on Computer Aided Design* 3 (1984), pp. 218-225.

[11] C.-L. E. Cheng. Risa: Accurate and Efficient Placement Routability Modeling, *ICCAD '94*, pp. 690-695.

[7] More important, such ε-changes may pose smaller implementation risk.

Test	Cells	Nets	White space	#Metal Layers	Placer	HPWL	WWL	Place time	Routed WL	Route time
1	11471	11828	24.3%	3	Industrial	2.80e6	3.22e6	182	**3.43e6**	223
+					UCLA Capo	2.68e6	3.05e6	269	**3.30e6**	293
					Capo-Fast	2.72e6	3.03e6	131	**3.38e6**	336
2	19832	22974	9.9%	6	Industrial	1.24e6	2.53e6	520	**2.49e6**	833
+					UCLA Capo	1.28e6	2.36e6	473	**2.16e6**	500
					Capo-Fast	1.31e6	2.12e6	270	**2.22e6**	502
3	20392	25634	14.2%	2	Industrial	5.93e6	7.70e6	414	**7.90e6**	613
+					UCLA Capo	5.60e6	7.18e6	471	**7.52e6**	579
					Capo-Fast	5.76e6	7.25e6	239	unroutable!	3840
4	25995	28603	8.7%	3	Industrial	1.08e7	1.36e7	427	**1.70e7**	5382
-					UCLA Capo	1.03e7	1.28e7	837	**1.79e7**	6346
					Capo-Fast	1.02e7	1.25e7	394	**1.78e7**	5558
5	33917	39152	29.4%	4	Industrial	5.89e6	7.29e6	990	7.90e6	3502
$-\infty$					UCLA Capo	5.73e6	6.88e6	1197	unroutable!	4196
					Capo-Fast	5.67e6	6.73e6	621	unroutable!	7355
6	35549	44121	0.1%	4	Industrial	9.67e6	1.23e7	1765	**1.18e7**	1120
+					UCLA Capo	9.30e6	1.14e7	1546	**1.11e7**	1050
					Capo-Fast	9.43e6	1.15e7	649	**1.17e7**	1055
7	42352	44490	29.3%	5	Industrial	3.71e7	4.77e7	981	**4.43e7**	624
o					UCLA Capo	3.62e7	4.55e7	1154	**4.45e7**	615
					Capo-Fast	3.48e7	4.51e7	657	**4.60e7**	701

Table 1: Comparison of our global placer, UCLA Capo (Spring 2000 version), in both regular ("UCLA Capo") and accelerated ("Capo-Fast") modes, with an industry tool (Fall 1999 version). We report half-perimeter pin-to-pin wirelength (HPWL) and weighted pin-to-pin half-perimeter wirelength (WWL) computed using the formulas of [11], and placer runtime for each testcase as well as the results of running an industrial routing tool on each placement produced. For the router, we report the final wirelength (if the router was successful) and router runtime. Runtimes are in user seconds on a 300MHz Sun Ultra10. Pluses indicate Capo gains on routed wirelength, minuses indicate losses and an "o" indicates a draw.

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

[13] A. E. Dunlop and B. W. Kernighan, "A Procedure for Placement of Standard Cell VLSI Circuits", *IEEE Transactions on Computer-Aided Design* 4(1) (1985), pp. 92-98.

[14] S. Dutt and H. Theny, "Partitioning Around Roadblocks: Tackling Constraints with Intermediate Relaxations", *ICCAD '97*, pp. 350-355.

[15] H. Eisenmann and F. M. Johannes, "Generic Global Placement and Floorplanning", *DAC '98*, pp. 269-274.

[16] C. M. Fiduccia and R. M. Mattheyses, "A Linear Time Heuristic for Improving Network Partitions", *DAC '82*, pp. 175-181.

[17] D. J. Huang and A. B. Kahng, "Partitioning-Based Standard Cell Global Placement With an Exact Objective", *ISPD '97*, pp. 18-25.

[18] S.W. Hur and J. Lillis, "Relaxation and Clustering in a Local Search Framework: Application to Linear Placement", *DAC '99*, pp. 360-366.

[19] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel Hypergraph Partitioning: Applications in VLSI Design", *DAC '97*, pp. 526-529. Additional publications and benchmark results for hMetis-1.5 are available at `http://www-users.cs.umn.edu/~karypis/metis/hmetis/`

[20] G. Karypis and V. Kumar, "Multilevel *k*-way Hypergraph Partitioning", *DAC '99*, pp. 343-348.

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

[22] T. Koide, M, Ono, S. Wakabayashi, Y. Nishimaru and N. Yoshida, "A New Performance Driven Placement Method with the Elmore Delay Model", *ASP-DAC '95*, pp. 405-412.

[23] K. Kozminski, "Benchmarks for Layout Synthesis – Evolution and Current Status", *DAC '91*, pp. 265-70.

[24] P. N. Parakh, R. B. Brown and K. A. Sakallah, "Congestion Driven Quadratic Placement", *DAC '98*, pp. 275-278.

[25] G. Sigl, K. Doll and F. M. Johannes, "Analytical Placement: A Linear or Quadratic Objective Function?" *DAC '91*, pp. 57-62.

[26] A. Srinivasan, K. Chaudhary and E. S. Kuh, "RITUAL: A Performance Driven Placement for Small-Cell ICs", *ICCAD '91*, pp. 48-51.

[27] P. R. Suaris and G. Kedem, "Quadrisection: A New Approach to Standard Cell Layout", *ICCAD '87*, pp. 474-477.

[28] W. Sun and C. Sechen, "Efficient and Effective Placements for Very Large Circuits", *ICCAD '93*, pp. 170-177.

[29] W. Swartz and C. Sechen, "Timing Driven Placement for Large Standard Cell Circuits", *DAC '95*, pp. 211-215.

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

[31] R. S. Tsay, E. Kuh, and C. P. Hsu, "PROUD: A Sea-Of-Gates Placement Algorithm", *IEEE Design & Test of Computers*, 1988, pp. 44-56.

[32] P. Villarrubia, G. Nusbaum, R. Masleid and P.T. Patel, "IBM RISC Chip Design Methodology", *ICCD '89*, pp. 143-147.

[33] P. Villarrubia, *personal communication*, 2000.

[34] J. Vygen, "Algorithms for Large-Scale Flat Placement", *DAC '97*, pp. 746-751.

[35] M. Wang and M. Sarrafzadeh, "Behavior of Congestion Minimization During Placement", *ISPD '99*, pp. 145-150.

[36] M. Wang, S. K. Lim, J. Cong and M. Sarrafzadeh, "Multi-way Partitioning Using Bi-partition Heuristics", *ASP-DAC 2000*, pp. 667-672.

[37] G. J. Wipfler, M. Wiesel, and D. A. Mlynski, "A Combined Force and Cut Algorithm for Hierarchical VLSI Layout, *DAC '83*, pp. 124-125.