Short Papers

Hypergraph Partitioning with Fixed Vertices

Charles J. Alpert, Andrew E. Caldwell, Andrew B. Kahng, and Igor L. Markov

***Abstract*—We empirically assess the implications of fixed terminals for hypergraph partitioning heuristics. Our experimental testbed incorporates a leading-edge multilevel hypergraph partitioner and IBM-internal circuits that have recently been released as part of the ISPD-98 Benchmark Suite. We find that the presence of fixed terminals can make a partitioning instance considerably easier (possibly to the point of being "trivial"); much less effort is needed to stably reach solution qualities that are near best-achievable. Toward development of partitioning heuristics specific to the fixed-terminals regime, we study the pass statistics of *flat* FM-based partitioning heuristics. Our data suggest that more fixed terminals implies that the improvements within a pass will more likely occur near the beginning of the pass. Restricting the length of passes—which degrades solution quality in the classic (free-hypergraph) context—is relatively safe for the fixed-terminals regime and considerably reduces the run times of our FM-based heuristic implementations. The distinct nature of partitioning in the fixed-terminals regime has deep implications: 1) for the design and use of partitioners in top-down placement, 2) for the context in which VLSI hypergraph partitioning research is pursued, and 3) for the development of new benchmark instances for the research community.**

***Index Terms*—Partitioning, physical design, placement, optimization.**

I. Introduction

Hypergraph partitioning research in very large scale integration (VLSI) computer-aided design (CAD) has been primarily motivated by the gate-level top-down placement context, which in modern ASIC design methodology can demand extremely efficient and high-quality solutions for netlist sizes exceeding one million vertices. New heuristics for hypergraph partitioning are typically evaluated in the context of *free hypergraphs*, where all vertices are free to move into any partition [2], [4]. *Every benchmark, and every benchmark result reported in the literature, is for the free-hypergraph context.* Even when input–output (I/O) pad locations are specified in the .vpnr or .yal source for early ACM/SIGDA benchmarks, the partitioning benchmarks (in .net/.are format; see [1]) do not indicate how these pads correspond to fixed vertices in partitions.

Our study is motivated by the following observation: *In top-down placement, the input to the partitioner is* **never** *a free hypergraph.* Rather, the input contains *fixed terminals* that arise from the chip I/O's or from the propagated terminals of other subproblems in the partitioning hierarchy [8], [14]. The number of these fixed terminals can be estimated from Rent's rule [7], [13], which states that in a layout with Rent parameter p, on average a block of C cells will have $T = k \cdot C^p$ propagated or external terminals. This corresponds to a partitioning instance of $C + T$ vertices, of which T are fixed. Here, k is a constant equal to the average number of pins per cell, and is approximately 3.5 for modern designs; Rent parameter values for modern designs have been estimated at around 0.68 [7], [15]. Table I shows the maximum block sizes below which we expect all blocks (in a design with Rent parameter p) to have a given percentage of their vertices fixed.[1] We observe that even rather sizable subblocks of the design can be expected to have a high proportion of fixed terminals.

TABLE I
Block Sizes Below Which the Expected Number of Fixed Vertices Due to Propagated Terminals Will Exceed a Specified Percentage (5%, 10%, or 20%) of the Total Number of Vertices in a Top-Down Placement When the Design Has Given Rent Parameter p. We Assume that the Average Pins per Cell in the Design Is $k = 3.5$

Rent Parameter	5%	10%	20%
$p = 0.60$	40992	7250	1281
$p = 0.65$	186943	25800	3561
$p = 0.70$	1413600	140250	13915

With this paper, we bring attention to the problem of partitioning with fixed terminals, and demonstrate that unique aspects of the fixed-terminals regime may require new partitioning heuristics. Hence, the nature of partitioning in the fixed-terminals regime can have deep implications: 1) for the design and use of partitioners in top-down placement, 2) for the context in which VLSI hypergraph partitioning research is pursued, and 3) for the development of new benchmark instances for the research community.

In Section II, we empirically assess the implications of fixed terminals for hypergraph partitioning heuristics using a leading-edge multilevel [3], [12] hypergraph partitioner and ISPD-98 circuit benchmarks released by IBM [1], [2]. We conclude that the presence of fixed terminals can make a partitioning instance considerably easier (possibly to the point of being "trivial"): less effort is needed to stably reach solution qualities near best-seen. Section III presents early studies aimed at developing partitioning heuristics specific to the fixed-terminals regime. We study the pass statistics of *flat* FM-based partitioning heuristics, and demonstrate that with more fixed terminals, the improvements in a pass are more likely to occur near the beginning of the pass. A heuristic that restricts the length of passes—which would degrade solution quality in the classic (free-hypergraph) context—is relatively safe for the fixed-terminals regime and considerably reduces runtime of our FM-based implementations. The existence of this heuristic demonstrates the need for improved heuristics that specifically exploit the fixed-terminals regime. Section IV concludes with directions for future work.

II. Effect of Fixed Terminals on Instance Difficulty

Our experimental studies address the following questions.

1) Can we identify and quantify the effects (with respect to runtime, solution quality relative to best-achievable, etc.) of fixed terminals in the instance on the performance of modern partitioning heuristics?
2) In particular, do partitioning instances with fixed terminals require less effort to "solve well" (using modern partitioning heuristics) than similarly-sized instances without fixed terminals?
3) Can we develop guidelines as to the effort (e.g., with respect to a multistart regime) required for modern partitioning heuristics to

Manuscript received June 1, 1999; revised October 7, 1999. This work was supported by a grant from Cadence Design Systems, Inc. This paper was recommended by Associate Editor D. Hill.

The authors are with the UCLA Computer Science Department, Los Angeles, CA 90095-1596 USA.

Publisher Item Identifier S 0278-0070(00)01790-5.

[1]This assumes that the blocks are in "Region I" of the Rent parameter fit [13].

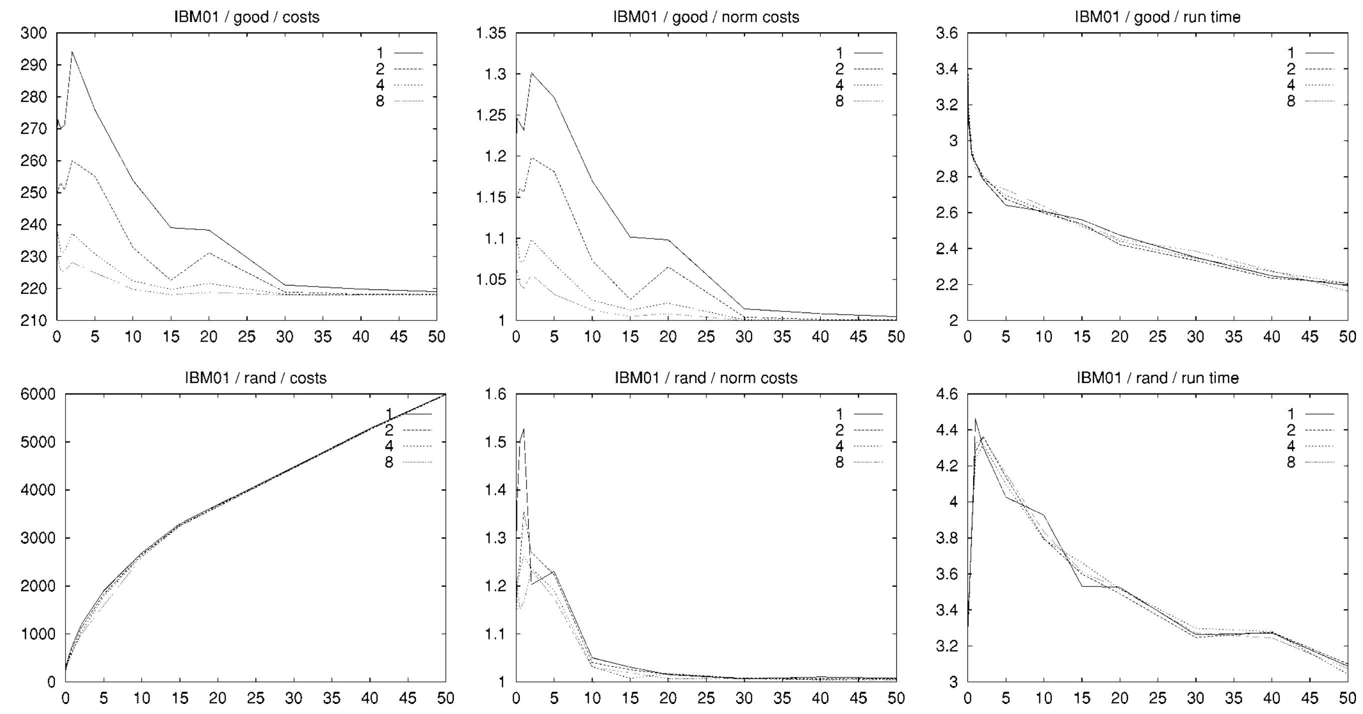

Fig. 1. Experimental results for IBM01 test case, actual cell areas, 2% balance tolerance. The four traces in each plot correspond to one, two, four, and eight starts of the multilevel partitioner. We report raw best solution costs (left column), normalized best solution costs (middle column), and total CPU times (right column) for both the "good" (upper row) and "rand" (lower row) regimes. In all plots, the given parameter is plotted against the percentage of fixed vertices in the instance.

achieve good partitioning solutions when a given proportion of vertices in the instance are fixed?

Partitioner: We use an internally developed partitioning engine that implements the *multilevel FM* approach described in [3] and [12]. Implementation details generally follow the parameters established in [3] (use of cluster-oriented iterative-improvement partitioner (CLIP) [9], heavy-edge matching, clustering ratio, etc.). The partitioning engine does not perform V-cycling as in [12], since we have determined that V-cycling is a net loss in terms of overall cost-runtime profile of our partitioner. The partitioning engine achieves solution quality and runtimes on a per-start basis that are somewhat better than those reported for ML_C [3] and $hMetis$ [12] in the 1998 paper of Alpert [2] and on Alpert's web page [1]. This is confirmed by the experimental data reported in the next section.

Test Data: We have run experiments with the IBM01 through IBM05 test cases from the ISPD-98 Benchmark Suite developed by Alpert [1], [2]. We use actual areas of cells, and a 2% balance constraint. Because the cell areas vary considerably in the IBM benchmarks (there are often individual cells that occupy several percent of the total area [1]), there is little point in doing unit-area studies for the real-life placement context. Moreover, tight balance constraints are more appropriate to the top-down cell placement application.

Experimental Protocol: In our experiments we choose to fix a subset of random vertices from the netlist. We either: 1) fix the chosen vertices independently into random partitions (**rand** in Figs. 1 and 2) or 2) fix the chosen vertices according to where they are assigned in the best min-cut solution we could find for the instance when no vertices were fixed (**good**). For each of the resulting four regimes we fix a number of vertices equal to 0%, 0.1%, 0.5%, 1.0%, 2.0%, 5%, 10%, 15%, 20%, 30%, 40%, and 50% of the total number of vertices in the instance.[2] We apply the multilevel CLIP FM engine noted in the previous subsection (using LIFO FM instead of CLIP FM results in very similar results to what we report in this paper). A single *trial* applies this partitioner to the given partitioning instance for one, two, four, or eight independent starts and returns both the best cutsize obtained and the number of CPU seconds used. (All CPU times are for a 140 MHz Sun Ultra-1 workstation running Solaris 2.6.) All of our data represent averages of 50 trials.

[2]We *incrementally* fix additional vertices, e.g., all vertices fixed at 1.0% are also fixed at 2.0%.

Experimental Results: Figs. 1 and 2 show detailed results for the IBM01 and IBM03 test cases, respectively. All data shown are for experiments where fixed vertices are chosen randomly from the set of all vertices in the instance.

- Each figure contains six plots, with four traces in each plot corresponding to one, two, four, and eight starts of the multilevel partitioning engine.
- The upper ("good") row of each figure presents plots for the regime where all fixed vertices are consistent with the best solution that we know for the unconstrained (no fixed vertices) instance. The lower ("rand") row of each figure presents plots for the regime where the fixed vertices are randomly assigned to partitions.
- The left two plots in each figure show the *raw solution costs* (best cutsize obtained with the given number of starts, averaged over 50 trials) versus the percentage of fixed vertices.
- Plots in the middle column in each figure show the *normalized solution costs* versus the percentage of fixed vertices. In the "good" regime, the normalization is to a single constant value (since all instances have fixed vertices consistent with the same good solution), so the shape of the traces is similar to the plot of raw solution costs. However, in the "rand" regime, the raw solution costs increase drastically with the percentage of randomly chosen/fixed vertices, and each percentage of fixed vertices corresponds to a distinct partitioning instance. Thus, for each instance in the "rand" regime, solution costs are normalized to the best solution cost seen over all $(1 + 2 + 4 + 8) \times 50 = 750$ starts of the multilevel partitioner for that instance.

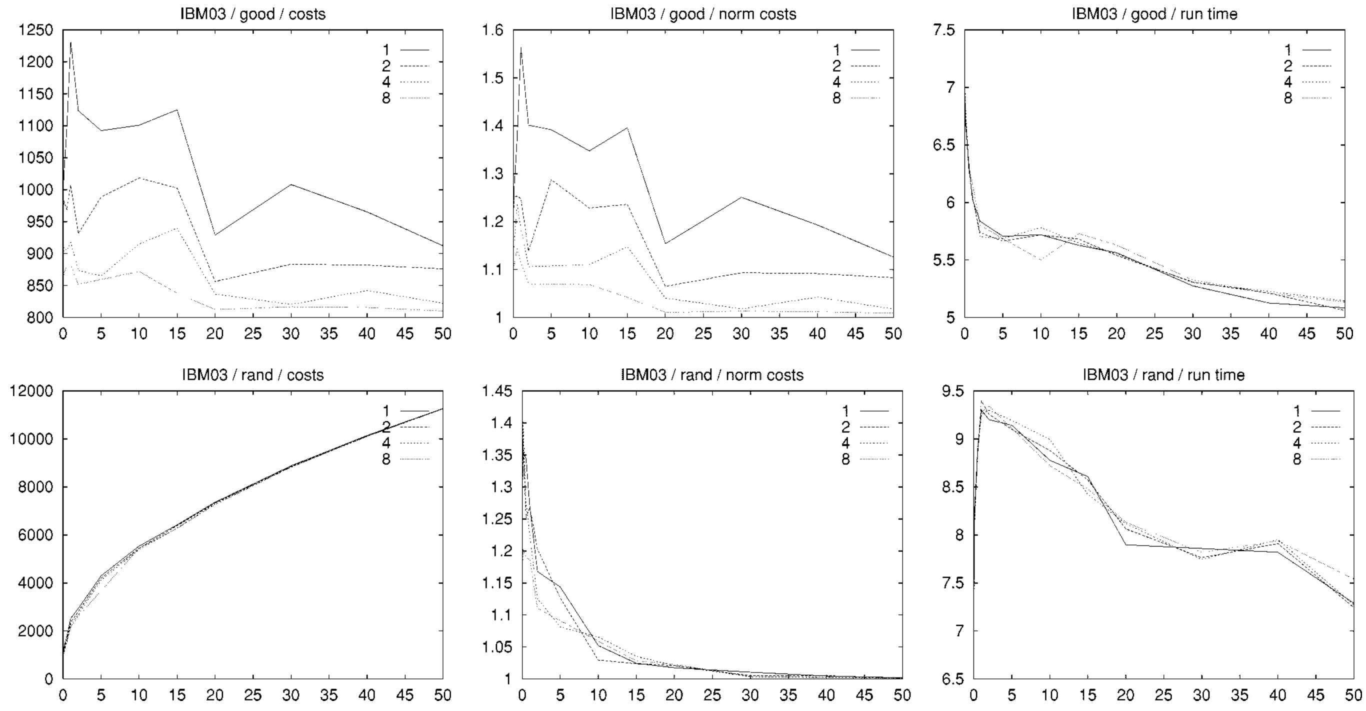

Fig. 2. Experimental results for IBM03 test case, actual cell areas, 2% balance tolerance. The four traces in each plot correspond to one, two, four, and eight starts of the multilevel partitioner. We report raw best solution costs (left column), normalized best solution costs (middle column) and total CPU times (right column) for both the "good" (upper row) and "rand" (lower row) regimes. In all plots, the given parameter is plotted against the percentage of fixed vertices in the instance.

- The right two plots in each figure show the *per-start CPU time* versus the percentage of fixed vertices.

We performed similar experiments where fixed vertices are chosen randomly from the set of identified I/O's (pads) in the netlist.[3] However, we do not discuss these results, for several reasons. First, the number of I/O's is typically very small (less than one percent of all vertices). Second, for those percentages of fixed vertices that could be chosen from I/O's we could find no difference in any experiment between fixing identified I/O's and fixing random vertices. Finally, for the vast majority of hierarchical block partitioning instances in top-down placement, the fixed terminals do not correspond to chip I/O pads anyway.

Results for other IBM benchmarks looked similar. From the Figures, we make the following observations.

- The raw solution costs[4] indicate that as more fixed vertices are (randomly) selected and assigned to partitions, the achievable solution cost increases rapidly. This addresses the first experimental question: the presence of fixed vertices matters.
- The normalized solution costs indicate that if the netlist has many terminals fixed in partitions (which is what we believe distinguishes real-life partitioning instances generated during top-down placement), then the partitioning problem is indeed "easy."
    - When 0% of the vertices are fixed in partitions, more starts (e.g., four or eight) are required for the average best cutsize to approach the value that the multilevel partitioner is capable of achieving for the given instance.
    - When larger percentages of the vertices are fixed in partitions, fewer starts (e.g., one or two) are required for the average best cutsize to approach the "good solution cost."
    - In the normalized traces, the curves are "flatter" (and there is less difference between the one-start and eight-start traces) as the percentage of fixed vertices increases.
    - In all of our experiments, an instance with 20% or more vertices fixed is essentially solvable to very high quality in one or two starts, i.e., further starts are unnecessary. This suggests that most hierarchical block partitioning instances in placement are easy; recall Table I from Section I.[5]
- Runtimes decrease substantially when the percentage of fixed vertices increases; this is expected since the partitioner has less freedom and a smaller number of movable vertices.
- Solution quality for "good" instances, and runtime for "rand" instances, are nonmonotonic in the percentage of fixed vertices. We suspect that this indicates "relatively overconstrained" instances where the inflexibility of the instance hurts the ability of the partitioner to find "trajectories to good solutions" more than it helps the partitioner by reducing the solution space. An interesting direction for future work is to attempt to demonstrate this effect. As discussed below, the data also suggest that current partitioning technology is not well-tuned to the fixed-terminals regime.

III. Toward Partitioners for the Fixed-Terminals Regime

A motivating observation is that in the absence of sufficient fixed terminals, FM may occasionally produce passes in which nearly every vertex is moved. Recall that during an FM pass, vertices are moved one at a time until each vertex has been moved; for bipartitioning, all vertices have been "flipped" when the end of the pass is reached. Then, the best solution found during the pass (i.e., best prefix of the move sequence) is restored. Any move "undone" in this process has essentially

[3]When the fixed vertices are chosen from pads in the netlist, the percentage is limited by the total number of pads, and we do not fix any further vertices.

[4]These raw solution costs suggest that our multilevel partitioner is (at least) on par with [3] and [12] in terms of solution quality.

[5]The benefit from additional starts decreases more noticeably in the "rand" regime than in the "good" regime. Since propagated terminals are not likely assigned to their ideal locations, the benefit from starts in the top-down placement context is likely somewhere between the "rand" and "good" portraits.

TABLE II
AVERAGE NUMBER OF PASSES PER RUN AND AVERAGE PERCENTAGE OF NODES MOVED PER PASS (EXCLUDING THE FIRST PASS), FOR 50 RUNS OF LIFO-FM

Testcase	Percent Fixed Terminals						
	0%	5%	10%	15%	20%	30%	50%
IBM01	12.0,20.5%	12.6,10.8%	9.9,9.1%	8.9,7.6%	8.0,4.8%	6.1,2.6%	4.6,0.8%
IBM02	9.6,12.4%	8.3,10.0%	8.1,6.1%	7.7,3.7%	6.7,2.7%	6.7,2.4%	6.4,3.2%
IBM03	10.4,6.8%	9.8,6.5%	8.3,5.7%	7.6,4.8%	6.4,3.3%	6.4,4.1%	6.0,3.7%
IBM04	12.9,8.3%	10.9,7.0%	9.9,4.4%	7.4,3.5%	8.1,2.6%	7.4,3.5%	6.1,1.0%
IBM05	32.6,37.0%	16.4,5.3%	13.0,4.7%	10.1,4.0%	8.6,3.5%	7.9,2.3%	5.3,1.5%

Partitions are allowed to deviate from exact bisection by 2%.

TABLE III
EFFECTS OF CUTTING OFF ALL PASSES (AFTER THE FIRST PASS) AT THE GIVEN MOVE LIMIT DURING LIFO-FM PARTITIONING

Test case	Max % to move	Percent Fixed Terminals					
		0%	5%	10%	20%	30%	50%
IBM01	nolimit	596.2(2.81)	1274.7(2.8)	1041.8(2.03)	513.8(1.51)	303.4(1.09)	247.0(0.724)
	50%	855.9(1.66)	1458.1(1.84)	1027.3(1.62)	555.2(1.28)	290.3(0.999)	244.6(0.672)
	25%	959.4(1.49)	1702.8(1.5)	1249.1(1.26)	521.7(0.98)	297.4(0.77)	245.0(0.593)
	10%	1233.6(1.47)	1811.4(1.39)	1435.3(1.18)	514.7(0.832)	298.0(0.665)	242.8(0.496)
	5%	1533.9(1.61)	2154.7(1.41)	1707.7(1.21)	631.5(0.845)	305.4(0.639)	247.4(0.462)
IBM03	nolimit	1929.7(4.46)	2544.7(4.03)	2867.0(3.31)	1709.0(2.22)	1509.8(2.1)	1713.3(1.8)
	50%	2327.9(3.61)	2831.0(3.22)	2870.5(2.62)	1666.8(2.01)	1524.5(2.11)	1811.2(1.94)
	25%	2160.5(2.56)	2896.8(2.75)	2643.2(2.32)	1729.3(1.72)	1653.0(1.88)	1412.7(1.45)
	10%	2250.4(2.35)	2967.4(2.25)	3054.5(1.95)	1403.2(1.37)	1456.1(1.41)	1728.1(1.37)
	5%	2198.7(2.17)	2985.4(2.05)	3100.4(1.98)	1634.4(1.3)	1711.6(1.4)	1633.3(1.25)
IBM05	nolimit	3455.6(18.7)	3324.3(9.53)	2461.4(7.21)	1972.1(4.48)	1854.3(4.07)	1793.5(2.16)
	50%	4726.8(7.96)	3254.0(6.43)	2486.2(4.94)	1979.4(3.83)	1857.6(3.28)	1793.3(2.23)
	25%	4954.4(6.31)	3336.3(5.99)	2432.0(4.37)	1970.3(3.11)	1856.2(2.57)	1793.2(1.75)
	10%	5234.6(6.31)	3528.6(4.72)	2545.3(3.93)	1991.8(2.45)	1850.6(2.25)	1793.0(1.48)
	5%	5730.7(6.51)	3676.0(5.99)	2576.1(4.06)	2010.5(2.45)	1849.8(2.06)	1794.0(1.47)

Partitions are allowed to deviate from exact bisection by 2%. Data is expressed as average cut (average CPU time). CPU seconds measured on a 300-MHz Sun Ultra-10.

been wasted. Without terminals, FM will occasionally produce passes in which almost no moves are wasted—the pass flips almost all vertices between partition zero and partition one. However, if there are sufficiently many vertices adjacent to fixed terminals, such a "near-flip" is very unlikely to be improving. Table II documents the average number of passes per run and the average percentage of vertices moved per pass: increasingly higher percentages of the moves in the FM passes are wasted as the proportion of fixed terminals increases. This strongly suggests that in the fixed-terminals regime, FM-style heuristics can profitably impose a hard cutoff on pass lengths.

Since the first FM pass traditionally begins with a random partitioning, many vertices will be moved, regardless of the number of fixed terminals. However, we may limit the number of moves per pass—after the first pass—in order to reduce overhead when the best solution found is near the beginning of the pass. Table III documents the effects on average cutsize and average CPU time for single LIFO FM starts, when FM passes are cut off after 50%, 25%, 10%, or 5% of the moves have been made. For instances without sufficient terminals, early stopping has a detrimental effect on solution quality, but with sufficient terminals no effect on solution quality is seen. In all cases, limiting the number of moves in a pass improves runtime without noticeable impact on solution quality. A surprising observation is that current partitioners appear to struggle when faced with only a small proportion (e.g., 5% or 10%) of fixed terminals. Because all terminals are fixed in a "good" location and because fixed terminals are added only to produce problems with a higher percentage of fixed vertices, any solution for the cases of 20% or 0% fixed is also feasible for the case of 10% fixed. The fact that the partitioner produces better results for both the 20% and the 0% cases than for the 10% case on IBM01 and IBM03 may point to a failing of current partitioners on those instances.

IV. TOWARD BENCHMARKS FOR THE FIXED-TERMINALS REGIME

Our experiments indicate that the nature of the partitioning problem may be changed by the presense of sufficiently many terminals. Since several effects cannot be presently explained, further research is required. To facilitate collaborative research on partitioning with fixed terminals, we propose a new type of benchmarks that capture sufficient information to make them a reasonable substitute to partitioning calls from running a top-down placer. We require the following features.

- Multiple partition geometries and capacities, fixed modules and terminal propagation, in virtually any combination. Sufficiently intelligent parsers will not require redundant information.
- Flexible balance constraints represented using absolute or relative (percentage) semantics.
- Straightforward facilities to represent "multibalanced" partitioning problems where each module supplies the same number ($k > 1$) of resource types.[6] A corresponding set of k capacities and tolerances must be specified for each partition. In a new "multi-area" file type, each "area" corresponds to a given resource type; this is a straightforward extension of the are file format with multiple module "areas" repeated on the same line.
- Flexible assignment of fixed terminals to partitions, which enables study of placement-specific partitioning objectives.[7] Terminals can be assigned to regions or to exact locations (via degenerate regions). Terminals can also be fixed in more than one

[6]A hypothetical example with $k = 3$ might include cell area, cell pin count, and cell power dissipation resource types—all of which must be evenly distributed between the partitions.

[7]For example, based on net bounding boxes and Steiner tree estimators, etc.

partition while still retaining their "atomic" nature, i.e., the multiple assignment is interpreted as an "or." For example, a propagated terminal can be fixed in the two left-side quadrants of a quadrisection instance, so that the partitioner is free to assign it to either left-side quadrant.

Detailed descriptions of new file formats are available in the Gigascale Silicon Research Center (GSRC) bookshelf for VLSI CAD algorithms [5] on the Web.

As a starting point in the development of a new benchmark suite, the IBM Corporation has supplied (x, y) location data for each cell and pad, corresponding to the actual placements of circuits in the ISPD-98 Benchmark Suite. From these placements, we develop partitioning instances with fixed terminals as follows.[8] A *block* is defined by a rectangular axis-parallel bounding box. An axis-parallel *cutline* bisects a given block. Each cell contained in the block induces a *movable vertex* of the hypergraph. Each pad adjacent to some cell in the block induces a zero-area *terminal* vertex of the hypergraph, fixed in the closest partition; adjacent cells not in the block similarly induce terminal vertices. From the placement of each IBMxx benchmark circuit, we extract four benchmark netlists IBMxxA–IBMxxD, each with two sets of terminal assignments (corresponding to vertical and horizontal cutlines). Each partitioning instance is named with the level at which it occurs (L0, L1, etc.) and the partitioning choices at higher levels which define it.[9] Obviously, there are many possible regions that could be defined by laying a slicing floor plan over the placement. We believe that our methodology for generation of benchmark instances allows us to select a manageable set of instances that reflects the top-down placement process while achieving a wide range of sizes and other instance characteristics. Parameters of the resulting instances are summarized in Table IV, showing the size of the largest cell in the instance as a percentage (Max%) of the total cell area. Also, both: 1) the number of "pads" and 2) the number of external nets (nets that are incident to at least one "pad") are given. Our construction creates more "pad" vertices in the hypergraph than there are external nets (the latter correspond to propagated terminals, as in [8]). This does not affect the partitioning problem since "pads" have zero areas. We have verified that the numbers of external nets in our benchmarks correspond reasonably to the statistics in Table I. The benchmarks can be downloaded from [5] together with information about best known solutions, partitioner run times for `hMetis-1.5.3` [12], etc.

V. Conclusions and Ongoing Work

We have empirically demonstrated a mismatch between the top-down placement context and current directions in VLSI CAD hypergraph partitioning research and benchmarking. We point out how easy the partitioning problem becomes when fixed terminals are present. We believe that there is a great deal of work remaining to be done in the area of extremely fast partitioning for the fixed-terminals regime, i.e., the real-world placement context.

Our early efforts have entailed per-pass analyses of *flat* FM-based partitioning heuristics, confirming that the presence of fixed terminals limits the improvements in a pass to moves made at the beginning of the pass. Imposing hard cutoffs on pass length—which degrades solution quality in the classical "free-hypergraph" context—is relatively safe in the presence of terminals, and considerably reduces runtime of our FM-based implementations.

[8]Because the circuits from the original ISPD-98 Benchmark Suite have been placed by different flows throughout IBM, the intermediate states of the placement process are not available as sources from which partitioning benchmarks with fixed terminals may be derived.

[9]For instance, L1_V0 is the left block of a top-level vertical bisection. See [5] and [6] for more details.

TABLE IV
PARAMETERS OF NEW BENCHMARKS WITH FIXED TERMINALS

Circuit	Cells	Pads	ExtNets	Nets	Pins	Max%
IBM01A_L0	12506	246	246	14111	50566	6.37
IBM01B_L1_V0	6388	1392	761	7384	27236	9.34
IBM01C_L1_V1	6121	1377	763	7370	26951	10.03
IBM01D_L3_C11-33	6739	2155	1227	7330	28661	9.54
IBM06A_L0	32332	166	166	34826	128182	13.56
IBM06B_L1_V0	13245	4360	1867	14786	58809	17.28
IBM06C_L1_V1	19094	4086	1851	21824	81997	16.11
IBM06D_L3_C11-33	10314	7553	3482	12438	55640	18.88
IBM09A_L0	53110	285	285	60902	222088	5.42
IBM09B_L1_V0	23461	28764	40003	47740	192358	5.49
IBM09C_L1_V1	29649	23211	40038	53040	204928	5.45
IBM09D_L3_C11-33	25099	27303	40137	49200	195924	5.50
IBM10A_L0	68685	744	744	75196	297567	4.80
IBM10B_L1_V0	25467	4971	3459	30072	115065	6.91
IBM10C_L1_V1	43231	5260	3506	48246	197408	5.78
IBM10D_L3_C11-33	26954	9376	6556	31529	129409	6.09
IBM11A_L0	70152	406	406	81454	280786	4.48
IBM11B_L1_V0	33506	4970	3323	40623	142760	6.40
IBM11C_L1_V1	36646	5036	3291	43935	152703	6.29
IBM11D_L3_C11-33	30971	9968	6068	36273	129724	6.92
IBM12A_L0	70439	637	637	77240	317760	6.43
IBM12B_L1_V0	38904	5233	3669	43660	179596	9.17
IBM12C_L1_V1	31544	5218	3610	36907	152863	9.90
IBM12D_L3_C11-33	27490	12411	8101	33215	142965	8.59
IBM13A_L0	83709	490	490	99666	357075	4.23
IBM13B_L1_V0	40582	6589	3578	49757	179870	6.09
IBM13C_L1_V1	43127	5895	3586	53246	196549	6.41
IBM13D_L3_C11-33	37193	12745	7754	45756	171864	6.42
IBM16A_L0	182980	504	504	190048	778823	1.89
IBM16B_L1_V0	99102	9083	5200	103816	432680	2.88
IBM16C_L1_V1	83884	10491	5211	91190	375500	2.81
IBM16D_L3_C11-33	65041	18326	8581	70899	300966	2.82
IBM17A_L0	184752	743	743	189581	860036	0.94
IBM17B_L1_V0	100763	10834	6684	106877	485785	1.58
IBM17C_L1_V1	84015	12290	6716	89052	409926	1.19
IBM17D_L3_C11-33	51714	23033	11444	58988	263516	2.10

An open and rather pragmatic issue is whether faster algorithms can be developed that exploit the presence of fixed terminals, e.g., in top-down placement. Further analysis of the effects of fixed terminals may be useful. In particular, it is not yet clear how to measure the "strength" of fixed terminals, or alternatively the "degree of constraint" in particular problem instances. While our experiments fix random terminals from known hypergraphs where most vertices have low degree, it is always possible to fix vertices of very high degree to yield qualitatively different problem instances with similar numbers of fixed terminals. Indeed, a bipartitioning instance with an arbitrary number/percent of fixed terminals can be represented by an equivalent instance with only two terminals, by clustering all terminals fixed in a given partition into one single terminal. For common partitioning heuristics, such a representation is likely to be just as easy or hard as the original instance; we, therefore, need to quantify the "degree of constraint" in an invariant way. Additional points of interest in partitioning with fixed terminals include: 1) determining whether multiway partitioning is as affected by fixed terminals and 2) confirming the existence of "relatively overconstrained" instances. Our new results on the topic will be posted at [5] as they become available.

References

[1] C. J. Alpert. Partitioning benchmarks for VLSI CAD community. [Online]. Available: HTTP: http://vlsicad.cs.ucla.edu/~cheese/benchmarks.html.

[2] ——, "The ISPD-98 circuit benchmark suite," in *Proc. ACM/IEEE Int. Symp. Physical Design*, Apr. 98, (see errata [Online]. Available: HTTP: athttp://vlsicad.cs.ucla.edu/~cheese/errata.html), pp. 80–85.

[3] C. J. Alpert, J.-H. Huang, and A. B. Kahng, "Multilevel circuit partitioning," in *Proc. ACM/IEEE Design Automation Conf.*, 1997, pp. 530–533.
[4] C. J. Alpert and A. B. Kahng, "Recent directions in netlist partitioning—A survey," *Integration*, vol. 19, pp. 1–81, 1995.
[5] A. E. Caldwell, A. B. Kahng, and I. L. Markov. GSRC bookshelf for VLSI CAD algorithms. [Online]. Available: HTTP: http://vlsicad.cs.ucla.edu/GSRC/bookshelf.
[6] ——, "Partitioning with terminals—A ~'new' problem and new benchmarks," presented at the ISPD-99.
[7] J. A. Davis, V. K. De, and J. D. Meindl, "A stochastic wire-length distribution for gigascale integration (GSI)—Part I: Derivation and validation," *IEEE Trans. Electron Devices*, vol. 45, pp. 580–589, Mar. 1998.
[8] A. E. Dunlop and B. W. Kernighan, "A procedure for placement of standard cell VLSI circuits," *IEEE Trans. Computer-Aided Design*, vol. 4, pp. 92–98, Jan. 1985.
[9] S. Dutt and W. Deng, "VLSI circuit partitioning by cluster-removal using iterative improvement techniques," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1996, pp. 194–200.
[10] C. M. Fiduccia and R. M. Mattheyses, "A linear time heuristic for improving network partitions," in *Proc. ACM/IEEE Design Automation Conf.*, 1982, pp. 175–181.
[11] D. J. Huang and A. B. Kahng, "Partitioning-based standard cell global placement with an exact objective," in *Proc. ACM/IEEE Int. Symp. Physical Design*, 1997, pp. 18–25.
[12] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning—Applications in VLSI design," in *Proc. ACM/IEEE Design Automation Conf.*, 1997, pp. 526–529.
[13] B. Landman and R. Russo, "On a pin versus block relationship for partitioning of logic graphs," *IEEE Trans. Comput.*, vol. C-20, pp. 1469–1479, Dec. 1971.
[14] P. R. Suaris and G. Kedem, "Quadrisection—A new approach to standard cell layout," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1987, pp. 474–477.
[15] D. Sylvester and K. Keutzer, "Getting to the bottom of deep-submicron," presented at the IEEE Int. Conf. Computer-Aided Design, Nov. 1998.

Slicing Floorplans with Range Constraint

F. Y. Young, D. F. Wong, and Hannah H. Yang

***Abstract*—In floorplanning, it is important to allow users to specify placement constraints. Floorplanning with preplaced constraint is considered recently in Murata *et al.* (1997) and Young and Wong (1998). In this paper, we address a more general kind of placement constraint called range constraint in which a module must be placed within a given rectangular region in the floorplan. This is a more general formulation of the placement constraint problem and any preplaced constraint can be written as a range constraint. We extend the Wong-Liu algorithm (1986) to handle range constraint. Our main contribution is a novel shape curve computation which takes range constraint into consideration. Experimental results show that the extended floorplanner performs very well and, in particular, it out-performs the floorplanner in [13] when specialized to handle preplaced modules.**

***Index Terms*—Floorplanning, placement constraint, physical design, simulated annealing, slicing floorplan.**

Manuscript received June 1, 1999; revised September 29, 1999. This work was supported in part by a grant from the Intel Corporation. This paper was recommended by Associate Editor D. Hill.

F. Y. Young is with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, New Territories, Hong Kong.

D. F. Wong is with the Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712-1188 USA.

H. H. Yang is wit the Intel Corporation, Hillsboro, OR 97124-5961 USA.

Publisher Item Identifier S 0278-0070(00)01800-5.

I. Introduction

Floorplan design is an important step in physical design of very large scale integration circuits. It is the problem of placing a set of circuit modules on a chip to minimize the total area and interconnect cost. In this early stage of physical design, most of the modules are not yet designed and, thus, are flexible in shape (soft modules), while some are reused modules and their shapes are fixed (hard modules). There are two kinds of floorplans: slicing and nonslicing. Many existing floorplanners are based on slicing floorplans [1], [2], [7], [10], [11]. There are several advantages of using slicing floorplans. Firstly, focusing only on slicing floorplans significantly reduces the search space which in turn leads to a faster runtime, especially when the simulated annealing method is used. Secondly, the shape flexibility of the soft modules can be fully exploited to give a tight packing based on an efficient shape curve computational technique [8], [9]. It has been shown mathematically that a tight packing is achievable [12] for slicing floorplans.

There are some interesting results in the direction of nonslicing floorplans recently. Two methods, sequence-pair [4] and bound-slice-line-grid (BSG) [6], have been proposed for placement of hard modules. The sequence-pair method has later been extended to handle preplaced modules [3] and soft modules [5]. In order to handle soft modules, it has to solve a mathematical programming problem to determine the exact shape of each module numerous times in the floorplanning process, and this results in long runtime.

In floorplanning, it is important to allow users to specify placement constraints. Three common types of placement constraints are preplaced constraint, boundary constraint, and range constraint. For preplaced constraint, we require a module to be placed exactly at a certain position in the final packing. In fact, the problem of floorplanning with obstacles can be solved by treating the obstacles as preplaced modules. This problem has been considered in both slicing and nonslicing floorplans [3], [5], [13]. For boundary constraint, we require a module to be placed along one particular side of the final floorplan: on the left, on the right, at the bottom, or at the top. This is useful when users want to place some specific modules along the boundary for input–output connections. This problem is considered recently in a slicing floorplanner [14]. In this paper, we consider the range constraint problem as an enhancement and extension of [13]. In this problem, we require a module to be placed within a given rectangular region in the final packing. This is indeed a more general formulation of the placement constraint problem and any preplaced constraint can be written as a range constraint by specifying the rectangular region such that it has the same size as the module itself. This less-restrictive constraint is useful in practice, but no previous work is reported on it before.

In this paper, we address this range constraint problem by extending the Wong–Liu algorithm [11]. Our main contribution is a novel shape curve computation which takes range constraint into consideration. When our algorithm is specialized to handle preplaced modules, we out-perform the floorplanner in [13]. Note that the floorplanner in [13] is also based on the Wong–Liu algorithm [11]. It extends the shape curve computation to understand preplaced constraint using the notion of reference point and a more complicated set of moves in the annealing process. In this paper, we use a simplier approach which uses only the original set of simple moves in [11]. Experimental results show that the extended floorplanner performs very well. The rest of the paper is organized as follows. We first define the problem formally in Section II. Section III presents our method to handle range constraint. Experimental results are shown in Section IV.