Hypergraph Partitioning for VLSI CAD: Methodology for Heuristic Development, Experimentation and Reporting*

Andrew E. Caldwell, Andrew B. Kahng, Andrew A. Kennings† and Igor L. Markov

UCLA Computer Science Department, Los Angeles, CA 90095-1596
†Cypress Semiconductor, Beaverton, OR 97008

Abstract

We illustrate how technical contributions in the VLSI CAD partitioning literature can fail to provide one or more of: (i) reproducible results and descriptions, (ii) an enabling account of the key understanding or insight behind a given contribution, and (iii) experimental evidence that is not only contrasted with the state-of-the-art, but also meaningful in light of the driving application. Such failings can lead to reporting of spurious and misguided conclusions. For example, new ideas may appear promising in the context of a weak experimental testbed, but in reality do not advance the state of the art. The resulting inefficiencies can be detrimental to the entire research community. We draw on several models (chiefly from the metaheuristics community) [5] for experimental research and reporting in the area of heuristics for hard problems, and suggest that such practices can be adopted within the VLSI CAD community. Our focus is on hypergraph partitioning.

1 Introduction

It is well-recognized that a contribution to the experimental literature should include (i) reproducible results and descriptions, (ii) an account of the key understanding or insight behind the contribution, and (iii) supporting evidence that is not only contrasted with the leading edge of knowledge, but also meaningful in light of the driving application. These precepts are, quite literally, at the heart of scientific discourse and discovery.

In the VLSI CAD context, where most research is application-driven and addresses metaheuristics for hard problems, failure to observe the above precepts can easily lead to reporting of erroneous or misleading conclusions. For example, a new algorithm idea may appear promising in the context of a weak experimental design or incomplete comparisons to previous work, but in reality does not advance the state of the art. On the other hand, useful ideas may be dropped due to the same causes. Inadequate descriptions can lead to irreproducible results, and tremendous amounts of wasted time. The resulting inefficiencies can be detrimental to the VLSI CAD community as a whole.

This paper centers on two intertwined issues: improved practices for experimental research in algorithms, and improved practices for scientific reporting in this domain. Our goal is to raise awareness of research and reporting methodologies by which "art" in experimental algorithms research can become more like "science". In what follows, Section 2 highlights three aspects of *research practice*: (i) correct abstraction of the application context and use model, (ii) achieving a robust testbed that supports correct experimental conclusions, and (iii) correct experimental design. We provide experimental evidence illustrating the magnitude of pitfalls inherent in the latter two aspects. Section 3 focuses on two aspects of *reporting practice*, namely, (i) reproducible descriptions of innovations and experiments, and (ii) experimental comparison of metaheuristics. Section 4 concludes the paper with open directions for future work. Throughout, we attempt to relate our ideas to those of several key works on experimental research and reporting for optimization heuristics (chiefly from the metaheuristics literature). While the benefits of principled research and reporting methodologies can never be formally justified, we believe that the supporting experimental evidence can be compelling.

*Research supported by a grant from Cadence Design Systems, Inc.

Hypergraph Partitioning in VLSI CAD

Our discussion will be grounded in the context of the *hypergraph partitioning* problem, which is formalized as follows. Given a hyperedge- and vertex-weighted hypergraph $H = (V,E)$, a *k-way partitioning* of V assigns the vertices to k disjoint nonempty partitions. The *k-way partitioning problem* seeks to minimize a given objective function $c(P^k)$ whose arguments are partitionings. A standard objective function is *cut size*, i.e., the number of hyperedges whose vertices are not all in a single partition; other objectives such as ratio-cut [37], scaled cost [11], absorption cut [36], etc. have also been proposed.

Constraints are typically imposed on the partitioning solution, and make the problem difficult. For example, the total vertex weight in each partition may be limited (*balance constraints*), which results in an NP-hard formulation [18]. Moreover, certain vertices can be fixed in particular partitions (*fixed constraints*). To achieve flexibility and speed in addressing various formulations, move-based heuristics are typically used, notably the Fiduccia-Mattheyses (FM) heuristic [17].[1] In this work, we address only FM-based 2-way partitioners.

2 Methodology for Research and Experiment Design

2.1 Abstracting the Context and Use Model

A seminal work from the metaheuristics literature, that of Barr et al. [5], notes the ways in which a heuristic method "makes a contribution". In particular, such a heuristic should be fast, accurate, robust, simple, high-impact ("solving a new or important problem faster and more accurately than other approaches"), generalizeable,

[1] Effective move-based heuristics for k-way hypergraph partitioning have been pioneered in such works as [29, 17, 8], with refinements given by [30, 32, 21, 31, 15, 3, 12, 20, 25, 16] and many others. A comprehensive survey of VLSI partitioning formulations and algorithms is given in [4]; a recent update on balanced partitioning in VLSI physical design is given by [22].

and/or innovative [5]. High-impact research in heuristics must necessarily address a problem that abstracts or otherwise reflects some real-world application. In this light, it is interesting to note a divergence between the VLSI CAD hypergraph partitioning literature and one of its (historically) most important driving applications.

Application: Top-Down Placement

A key driver for hypergraph partitioning research in VLSI CAD has been the top-down global placement of standard-cell designs. In this application, scalability, speed and solution quality are all important objectives. Careful consideration of these issues with respect to the type of instances typically encountered, as well as the use model, helps bound the space of useful algorithmic solutions.

Nature of instances. Depending on the specific VLSI design application, a partitioning instance may have directed or undirected hyperedges, weighted or unweighted vertices, etc. Reporting has centered on the ACM/SIGDA benchmarks (www.cbl.ncsu.edu/benchmarks). Alpert [2] noted that many of these circuits no longer reflect the complexity of modern partitioning instances, particularly in VLSI physical design; 18 larger benchmarks arising in the layout flow of internal IBM designs were released in early 1998 [1]. Salient attributes of real-world inputs:

- size: number of vertices up to one million or more (all instance sizes equally important)
- sparsity: number of hyperedges very close to the number of vertices; average vertex degrees typically between 3 and 5 for device-, gate- and cell-level instances, with higher averages in block-level design
- average net sizes typically between 3 and 5
- small number of extremely large nets (e.g., clock, reset).
- wide variation in vertex weights (cell areas) due to the drive range of deep-submicron cell libraries, presence of complex cells and large macros in the netlist, etc.

Use model. A modern top-down standard-cell placement tool might perform timing- and routing congestion-driven recursive min-cut bisection of a cell-level netlist to obtain a "coarse placement", which is then refined into a "detailed placement" by stochastic hill-climbing search. The *entire* placement process in currently released commercial tools requires approximately 1 CPU minute per 6000 cells on a 300MHz Sun Ultra-2 uniprocessor workstation with adequate RAM. The implied partitioning runtimes are on the order of 5 CPU seconds for netlists of size 25,000 cells, and under one CPU minute for netlists of size 750,000 cells [13]. Quite possibly, experimental validation of heuristics in the wrong runtime regimes (say, hundreds of CPU seconds for a 5000-cell benchmark, without extenuating justifications) has no practical relevance. We also observe that in top-down placement, almost all hypergraph partitioning instances have many vertices fixed in partitions due to terminal propagation [14, 35] or pad locations. Separate work of ours [9] suggests that the presence of fixed terminals fundamentally changes the nature of the partitioning problem from that embodied in the "unfixed" benchmarks currently studied. We believe that these two aspects of the use model for hypergraph partitioning suggest heuristics that are optimized for speed and "easy" instances; such heuristics may turn out to be different from those in the current literature.

2.2 Robust Experimental Testbed

To identify algorithmic improvements at the leading edge of heuristic technology, it is critical to evaluate proposed algorithm improvements not only against the best available implementations, but also *using a competent implementation*. In our experience, proposed "improvements" often look good if applied to weak or slow implementations, but may actually worsen strong or fast implementations (cf., e.g., the maxims "Do make it fast enough" and "Do measure CPU time" given by Gent et al. [19]). An incorrectly implemented testbed can lead to incorrect conclusions, which then waste the research community's efforts to reproduce the improvement that purportedly results from the new technique.

In the following, we illustrate the danger of irreproducible and possibly meaningless experimental conclusions that stem from a poorly implemented or understood partitioning testbench. Our intest is to raise awareness of a very much underestimated root cause of a poor testbench, namely, the failure to understand *implicit implementation decisions* that can dominate quality/runtime tradeoffs (cf. "Do report important implementation details" [19]). A corollary is that researchers must take pains to identify and clearly describe such implicit decisions in order for results to be reproducible.

With respect to the Fiduccia-Mattheyses algorithm, we believe that many implementation pitfalls correspond to *implicit decisions* – underspecified features and ambiguities in the original algorithm description that must be resolved in any particular implementation. Examples include the following.

- *Tie-breaking* in choosing the highest gain bucket when selecting the next move to make. When the FM gain structure is implemented such that available moves are segregated, typically by source or destination partition, there can be more than one nonempty highest-gain bucket. Then, if the balance constraint is anything other than "exact bisection", it is possible for all the moves at the heads of the highest-gain buckets to be legal. The FM implementer must choose a method for dealing with this situation. Below, we contrast three approaches: (i) choose the move that is not from the same partition as the last vertex moved (**"away"**); (ii) choose the move in partition 0 (**"part0"**); and (iii) choose the move from the same partition as the last vertex moved (**"toward"**).
- Whether to update, or skip the updating, when the *delta gain* of a move is zero. When a vertex x is moved, the gains for all vertices y on nets incident to x must potentially be updated. In all FM implementations that we know of, this is done by going through the incident nets one at a time, and computing the changes in gain for vertices y on these nets. A straightforward implementation computes the change in gain ("delta gain") for y by adding and subtracting four cut values for the net under consideration, and immediately updating y's position in the gain container. Notice that sometimes the delta gain can be zero. Below, we show the effect of the implicit decision whether to reinsert a vertex y when it experiences a zero delta gain move (**"AllΔ_{gain}"**), or whether to skip the gain update (**"Nonzero"**). The former will shift the position of y within the same gain bucket; the latter will leave y's position unchanged. The effect of zero delta gain updating is not immediately obvious.[2]
- *Tie-breaking* on where to attach new element in gain bucket,

[2] The gain update method presented in [17] has the *side effect* of skipping all zero delta gain updates. However, this method is both netcut- and two-way specific; it is by no means certain that the FM implementer will find analogous solutions for k-way partitioning with a general objective.

ALGORITHM		TESTCASE		
Updates	Bias	**ibm01**	**ibm02**	**ibm03**
Flat LIFO FM				
All Δ_{gain}	Away	1204/**1885**	723/**3256**	3531/**4389**
All Δ_{gain}	Part0	1333/**1909**	558/**2440**	2998/**4166**
All Δ_{gain}	Toward	485/**1023**	342/**1274**	3074/**3939**
Nonzero	Away	333/**639**	271/**551**	1597/**2838**
Nonzero	Part0	293/**660**	321/**573**	1684/**2938**
Nonzero	Toward	319/**607**	291/**543**	1592/**2843**
Flat CLIP FM				
All Δ_{gain}	Away	373/**842**	292/**1841**	1664/**3623**
All Δ_{gain}	Part0	361/**772**	384/**1499**	1336/**3543**
All Δ_{gain}	Toward	327/**615**	333/**945**	1482/**3066**
Nonzero	Away	323/**542**	300/**574**	1531/**2689**
Nonzero	Part0	282/**556**	319/**582**	1237/**2732**
Nonzero	Toward	339/**528**	297/**562**	1201/**2504**
ML LIFO FM				
All Δ_{gain}	Away	216/**289**	280/**433**	744/**958**
All Δ_{gain}	Part0	217/**289**	266/**429**	795/**957**
All Δ_{gain}	Toward	216/**289**	266/**423**	805/**971**
Nonzero	Away	219/**287**	279/**432**	786/**969**
Nonzero	Part0	216/**282**	276/**421**	764/**952**
Nonzero	Toward	217/**276**	266/**419**	785/**959**
ML CLIP FM				
All Δ_{gain}	Away	219/**283**	285/**428**	807/**960**
All Δ_{gain}	Part0	216/**289**	266/**441**	790/**969**
All Δ_{gain}	Toward	218/**284**	266/**425**	782/**953**
Nonzero	Away	217/**283**	266/**414**	760/**957**
Nonzero	Part0	220/**285**	266/**447**	749/**934**
Nonzero	Toward	216/**282**	275/**433**	780/**959**

Table 1: Best and average cuts in partitioning with **actual** areas and **2%** balance tolerance, over 100 independent runs.

e.g., LIFO versus FIFO versus random (this has been addressed in the literature by [21]).[3]

- *Tie-breaking* when selecting the best solution encountered during the pass, e.g., choose the first such solution, the last such solution, or the one that is furthest from violating balance constraints.

Table 1 reports pairs of form (minimum cut **/ average cut**) for bipartitioning with four partitioner variants on IBM test cases [2] with actual cell areas, 2% balance constraint (i.e., partition sizes constrained to be between 49% and 51% of total cell area). From the data, we see the effects of just these two implicit decisions:

- The average cutsize for a flat partitioner can increase by startling amounts if the worst combination of choices is used instead of the best combination. Such effects far outweigh the typical solution quality improvements reported for new algorithm ideas in the partitioning literature.

- Stronger optimization engines (order of strength: ML CLIP > ML LIFO > flat CLIP > flat LIFO) can tend to decrease the "dynamic range" for the effects of implementation choices. This is actually a danger: e.g., developing a multilevel FM package may hide the fact that the underlying flat engines are badly implemented. At the same time, the effects of a bad implementation choice can still be apparent even when that choice is wrapped within a strong optimization technique (e.g., ML CLIP).

[3] In [21], where the authors show that inserting moves into gain buckets in LIFO order is much preferable to doing so in FIFO order (also a constant-time insertion) or at random. Since the work of [21], all FM implementations that we are aware of use LIFO insertion.

Tolerance	Algorithm	ibm01	ibm02	ibm03
02%	Reported LIFO	450/2701	648/12253	2459/16944
02%	Our LIFO	366/594	301/542	1588/2688
10%	Reported LIFO	270/486	313/3872	1624/12348
10%	Our LIFO	244/445	266/405	1057/1993

Table 2: Comparison of our LIFO FM results ("Our LIFO") against other LIFO FM results [2] ("Reported LIFO"), for ISPD98 benchmark suite cases. Results shown are minimum/average cutsizes obtained over 100 independent single-start trials, with 2% and 10% balance constraints and actual cell areas.

Examples of such effects are visible throughout the literature (e.g., compare "FM" results for exact unit-area bisection of MCNC test cases). A recent example is found in [2], which reports LIFO FM results on the new ISPD98 partitioning benchmark suite. Table 2 shows that the difference in solution quality obtained by our "LIFO FM" implementation and the "LIFO FM" implementation of [2] is substantial; this supports our claim that silent implementation choices can swamp the typical claimed improvements of algorithm innovations. Taxonomies of other (hidden) implementation decisions are given by, e.g., Hauck and Borriello [20], who note the effect of initial solution generation, and by Caldwell et al. [10]. The latter work also points out the need for a flexible, highly modular testbench architecture to facilitate algorithm experimentation and correct pruning of implementation options.

2.3 Experiment Design

Since even the most careful software implementation can miss some insight, impactful research must also entail rigorous experimental comparisons to leading-edge approaches ("Do check your health regularly" [19]). This ensures that a new technique improves solution quality in an absolute sense, rather than in just a relative sense vis-a-vis some potentially low-quality "internal" implementation. (A necessary attribute of experimental comparisons is that they are "apples to apples".)

Barr et al. [5] observe that research about heuristics is valuable if it is "revealing", i.e., "offer[s] insight into general heuristic design or the problem structure by establishing the reasons for an algorithm's performance and explaining its behavior". To this end, a research experiment should be driven by a clear statement of "the questions to be answered and the reasons that experimentation is required". Maxims listed by Gent et al. [19] under the heading of experiment design include "Do measure with many instruments", "Do vary all relevant factors", "Don't change two things at once", "Do collect all data possible", etc. – admittedly obvious concepts on one hand, but all too uncommonly observed on the other hand. In this subsection, we illustrate how imperfect experiment design can detract from understanding and the value of the research result. Our illustration points out that deep understanding of heuristics may be required even to formulate the experiments that drive and verify algorithm innovation.

We consider the CLIP algorithm of [15], which has been enabling within a recent multilevel partitioner implementation [3]. We make two observations.

- First, FM-based partitioners typically look at only the first move in a bucket. We believe this is partly for speed, and partly as a historical legacy from partitioners being tuned for unit-area, exact-bisection benchmarking. (Note that moves are examined in priority order, so the first legal move found is the best.) If the first move is legal it is made; if the move is not

Tolerance	Algorithm	ibm01	ibm02	ibm03
02%	Reported CLIP	471/2456	1228/12158	2569/16695
02%	Our CLIP	329/485	298/472	797/1635
10%	Reported CLIP	246/462	439/4163	1915/9720
10%	Our CLIP	237/424	266/406	675/1325

Table 3: Comparison of our CLIP FM implementation ("Our CLIP") and the CLIP FM reported in [2] ("Reported CLIP"), for ISPD98 benchmark suite cases. Our CLIP FM does not insert cells with area greater than the balance constraint into the gain structure. Results shown are minimum/average cutsizes obtained over 100 independent single-start trials, with 2% and 10% balance constraints and actual cell areas.

legal, the entire bucket (or perhaps even every bucket for that partition) is skipped. This strategy is reasonable since it is very time-consuming to traverse a bucket's entire list, hoping that one of the nodes in it will be legal.

- Second, the CLIP algorithm begins by placing all moves into 0-gain buckets. This is because CLIP chooses moves by their cumulative delta gain (i.e., the "actual gain" minus the initial gain), and initially every move has a cumulative delta gain of 0. Hence, if the move at the head of each bucket at the beginning of a CLIP pass is not legal, the whole pass terminates without making any moves. Furthermore, even if the first move is legal, CLIP is still likely to get stuck if the moves very soon afterwards in both buckets are illegal. This is because there will not be enough time for the moves to "spread out", and nearly all will still be in the zero-gain bucket. We call this the *corking effect* in CLIP: the large node at the head of the list (bucket) acts as a *cork*.

Traces of CLIP executions show that corking actually occurs fairly often, particularly with the more modern ISPD98 actual-area benchmarks. CLIP places the cells with highest (total initial) gain at the heads of the zero gain buckets at the beginning of the pass. Particularly in a random solution, the cells with the highest gain will tend to be the cells of highest degree, which are also the cells with greatest area. Any number of simple techniques can be conceived to address this problem. For example:

- do not place cells that have area greater than the balance tolerance into the gain structure at the begining of the pass (this technique actually benefits all FM variants, and has essentially zero overhead); or
- look beyond the first move in a bucket, if the first move is illegal (we find this to be too time-consuming, and it moreover appears to have a harmful effect on solution quality).

Table 3 shows that the trivial first approach ("Our CLIP") can lead to noticeable performance differences versus other CLIP implementations. We believe that the susceptibility of the original CLIP algorithm to corking is fundamentally due to experiment design: corking is masked by the tendency to compare partitioners according to unit-area bisection results (cf. "Do understand your problem generator" [19]).[4] This suggests that the research community could benefit from increased vigilance regarding the testing of proposed algorithms within actual use models. Algorithms should ideally be tested on the full range of applicable problems; more generally, any prospective advance in algorithm technology should be evaluated in a range of contexts, and within any particular context should conform to all the requirements imposed by that context.[5]

3 Methodology for Reporting

3.1 Algorithm Description

As noted earlier, Gent et al. propose the maxim, "Do report important implementation details". Section 2.2 above showed that algorithm innovation depends on having the strongest possible baseline implementation testbed. Attaining such a testbed is not easy since many implicit implementation decisions must be correctly identified and made – and at a minimum, the list of reported algorithm details should include these implementation decisions. In our experience, the effects of such implementation elements as randomizers, initial solution generators, tie-breaking decisions, etc. can be very large – particularly when metaheuristic comparisons are made (see the next subsection). We suggest that with the maturation of the VLSI CAD field, *benchmark algorithm implementations* (available in source code form) will become at least as important as *benchmark data* in driving cost-effective research and development.

3.2 Comparison of Metaheuristics

Most papers in the VLSI CAD partitioning literature report average and best solution quality obtained over some fixed number of independent starts of a given heuristic (e.g., 20 or 100 starts). This reporting style can obscure the quality-runtime tradeoff curve, which is very unpredictable given widely varying problem sizes, constraints and hypergraph topologies. Furthermore, for the motivating context of top-down standard-cell placement, there is no realistic use model in which a partitioner will be called 20 times (let alone 100 times) for a given instance. Realistic runtime regimes support at most a few starts of the fastest partitioners of which we are aware.

In the metaheuristics and, e.g., INFORMS communities, experimental reporting methodology for metaheuristics has been the subject of much discussion and consensus-building. Barr et al. [5] describe a number of standard reporting styles; the most popular is the "best-so-far (BSF) curve", which plots the solution cost that the algorithm is expected to achieve in a multistart regime, versus the given CPU time budget τ. As noted by Schreiber and Martin [33] [34], *speed-dependent ranking* methodologies for the multistart regime can be based on variants of BSF evaluation. In particular, such methodologies use the distribution of c_τ, the best solution cost achieved in time τ.[6] Different heuristics can be ranked based on the mean values of c_τ, based on the probabilities that $c_\tau = C_0$, etc. This yields a useful *ranking diagram* diagnostic that depicts regions of, e.g., (instance size, CPU time) dominance for each of the heuristics being compared. Statistical analyses (e.g., significance tests) are also recognized as helpful in evaluating the significance of solution cost variation in diverse circumstances; Brglez has recently pointed this out, along with effects of randomizations, in the VLSI CAD literature [7].

We support the comparison of heuristics using methods similar in spirit to the BSF curves presented in [5], or the speed-dependent

[4] The well-defined nature of certain CAD problems (notably partitioning) and the strong push for even small improvements in quality of results may lead to a tendency towards fragile algorithm innovations. The fact that CLIP corking was not previously realized is due to testing of algorithms on an incomplete set of data; the older MCNC test cases lack large cells, and have historically been used in "unit-area" mode.

[5] In particular, no algorithm in the VLSI CAD partitioning literature has *a priori* restricted its domain of application to, e.g., only unit-area bisection. Thus, it is eminently reasonable to evaluate VLSI CAD partitioning algorithms on non-unit area instances, as long as such instances arise in the motivating application domains.

[6] Based on the average runtime of a single start of a heuristic, a given time bound τ can be converted to a bound on the number of starts.

rankings of [33] [34]. One observation is that it is essential to use actual CPU time as an axis of comparison, as opposed to coarser-grain quanta such as “number of starts”. This is because many advanced metaheuristics, including VLSI multilevel partitioning heuristics, do not necessarily use independent starts. For example, pruning (early termination of starts that appear unpromising relative to previous starts) can be applied, along with techniques such as V-cycling [25, 26] that are invoked only for the best result of several starts (this implies that sampling methods [33] [34] cannot be used). In comparing algorithms, we say that a particular (solution cost , runtime) performance point A is *dominated* by another performance point B if and only if B has both lower cost and lower runtime than A. (In other words, no one would ever choose to run configuration A over configuration B.) We then define the *non-dominated frontier* of (solution cost , runtime) performance points to be the set of all such points that are not dominated by any other points; this is exactly the Pareto set in multi-objective optimization. The non-dominated frontier (Pareto set) of performance points obtained from multiple heuristics allows the reader to easily see which heuristic is preferable for a given runtime regime, how much cost improvement can be obtained for a given increment of runtime, etc.

Appropriate Comparisons

Advances in the VLSI partitioning literature are traditionally reported with respect to a collection of benchmark instances. It is critical that the instances be appropriately chosen. For the top-down placement context, a wide range of instance sizes best emulates the actual use model. It is also important that the benchmarks be as up-to-date as possible. In particular, we believe that the new ISPD98 benchmark suite (parameters given in [2] and on the Web [1]) is now much more relevant than the MCNC cases.[7]

To match the top-down placement use model, we believe that actual-area partitioning is the most relevant test of a partitioner. Traditional balance constraints of 2% (partition areas between 49% and 51% of total cell area) and 10% (partition areas between 45% and 55% of total cell area) are both relevant, since vertical cutlines can be located rather continuously in the design.[8] As noted above, practical runtime budgets are very tight, even for large instances. Finally, results should be compared against the leading edge.

Example Metaheuristic Evaluation: hMetis-1.5

As an example, performance evaluation of the leading available partitioner (the hMetis-1.5 executable available on the Web from the University of Minnesota [28, 26]) might be reported as follows. A similar performance evaluation for our own internally developed partitioner is available at our website, `http://vlsicad.cs.ucla.edu/`.

- We run hMetis-1.5 using precisely its default configurations (cf. the description of “shmetis” in [27]), and vary only the number of starts. We run hMetis-1.5 using number of starts equal to 1, 2, 4, 8, 16 and 100 (note that 100 starts is the standard way in which hMetis solution quality had been presented in the past [2], and is in some sense a limit of practical interest). hMetis-1.5 will V-cycle the best result among these starts. For each number of starts (corresponding to “Configurations” 1 - 6, respectively) of hMetis-1.5, we execute hMetis-1.5 with that number of starts 50 different times, and record the average best cutsize and the average CPU time. All CPU times are normalized to 200MHz Sun Ultra-2 CPU times.[9]
- We run the partitioner on the IBM01-06, IBM10, IBM14 and IBM18 benchmarks, using actual cell areas and both 10% and 2% balance tolerance. Missing benchmarks are omitted in this work only because of the very large CPU burden of running so many trials for each configuration of each test case (we run the equivalent of nearly 10,000 starts for each test case).

Circuit	Configuration					
	1	2	3	4	5	6
ibm01	265.7/6.4	264.1/8.2	248.0/12.0	246.9/19.7	236.8/42.7	224.5/216.5
ibm02	318.5/11.6	315.1/14.8	305.1/21.0	304.1/35.0	287.8/77.9	269.5/369.0
ibm03	894.8/18.3	873.9/22.1	852.5/32.7	862.3/51.5	797.9/116.9	755.3/518.1
ibm04	564.5/15.8	542.2/21.4	536.0/31.3	532.4/51.7	523.7/111.2	516.2/549.7
ibm05	1768.1/26.1	1746.3/30.8	1742.6/44.8	1740.6/77.6	1735.2/165.9	1730.5/836.0
ibm06	719.0/27.6	668.1/33.2	650.8/46.2	597.1/70.5	570.8/163.8	543.9/675.9
ibm10	1328.3/65.9	1283.2/81.7	1217.3/113.5	1203.7/176.7	1142.9/393.7	1108.4/1739.0
ibm14	2113.4/163.0	1966.6/195.0	1906.9/281.0	1853.9/447.0	1850.4/966.6	1770.0/4348.9
ibm18	1858.2/248.2	1812.9/294.9	1750.9/436.3	1719.7/740.1	1681.2/1599.6	1666.2/7529.9

Table 4: Evaluation of hMetis1.5 [26] [27] for IBM test cases from the ISPD98 benchmark suite [2]. Solutions are constrained to be within 2% of bisection (partitions must contain between 49% and 51% of total cell area). Data expressed as (average cut / average CPU time), with the latter normalized to CPU seconds on a 200MHz Sun Ultra-2.

Circuit	Configuration					
	1	2	3	4	5	6
ibm01	258.6/6.0	252.5/8.3	246.1/11.7	242.2/19.7	227.3/41.9	216.6/209.2
ibm02	280.4/13.0	277.7/16.4	276.6/22.3	278.8/33.4	273.5/75.5	272.7/337.3
ibm03	787.8/17.3	781.3/21.9	776.0/29.6	749.3/49.0	736.6/108.4	693.2/490.8
ibm04	511.5/18.2	510.7/23.6	492.1/34.1	475.7/55.1	455.8/120.9	441.3/553.9
ibm05	1744.7/30.2	1742.2/32.0	1734.2/45.1	1724.1/75.1	1719.3/162.5	1713.5/790.4
ibm06	386.5/26.1	383.5/29.5	377.8/38.6	373.0/61.4	367.8/134.6	367.0/630.2
ibm10	817.0/52.1	805.9/66.3	799.2/95.5	778.6/154.4	770.9/342.0	760.6/1606.7
ibm14	1871.9/154.3	1810.4/182.5	1640.9/270.8	1593.5/421.6	1566.0/950.3	1527.5/4216.6
ibm18	1631.3/192.3	1632.4/272.4	1559.2/370.5	1533.0/612.5	1527.7/1336.9	1521.6/6479.2

Table 5: Evaluation of hMetis1.5 [26] [27] for IBM test cases from the ISPD98 benchmark suite [2]. Solutions are constrained to be within 10% of bisection (partitions must contain between 45% and 55% of total cell area). Data expressed as (average cut / average CPU time), with the latter normalized to CPU seconds on a 200MHz Sun Ultra-2.

Partitioning results are reported in Tables 4 and 5 so as to reveal the runtime-quality tradeoff in the region of practical interest – in addition to the traditional evaluation (100 starts). Data omitted for readability in this medium include the standard deviations and other descriptors of the distributions of all numbers; any more flexible presentation medium for our results (e.g., a webpage) should contain such information.

4 Conclusions

Despite much progress over the past several decades, the VLSI CAD (hypergraph partitioning) research community can still benefit from improved understanding of the fundamental heuristics upon

[7] The MCNC cases are small and lack nodes with large degree or large area. In contrast, the ISPD98 cases have up to 210,341 cells and also include many large macro-cells.

[8] Due to the discrete nature of cell rows, horizontal cutlines do require tighter balance constraints, or else a partitioning cleanup step to “snap” into row-compatible partition sizes.

[9] Due to the extremely large amount of CPU required to perform our experiments, some experiments were run on 110MHz Sun Sparc-5’s and on 300MHz Sun Ultra-10’s. Runtime conversion factors were computed on an instance-specific basis by comparing runtimes for hMetis-1.5 execution on different machines but with identical random seeds.

which new methods are continually developed. We suggest that technical contributions should more diligently pursue reproducible results and descriptions, as well as deeper understanding of interactions between metaheuristic choices. On the research side, experimental testbeds and experimental designs should enable careful contrast to the leading edge of the field, and should also be relevant to the complete application and use model context. For hypergraph partitioning, the examples of "implicit implementation decisions" and the "corking" effect in CLIP FM [15], we have illustrated the possibility of spurious conclusions for which the "error" can swamp claimed solution quality improvements due to algorithm innovation. We further suggest that a culture of benchmark metaheuristic implementations may in the future be as enabling to the field as a culture of benchmark data. On the reporting side, complete descriptions are necessary, along with more principled (and standardized) methodologies for comparison of metaheuristics. Here, useful guidance can be obtained from the metaheuristics literature. We provide illustrative fragments from the evaluation of one leading metaheuristic, the hMetis-1.5 partitioner from the University of Minnesota [28, 26].

Our ongoing research seeks to improve our basic understanding of VLSI partitioning (e.g., we believe that the effects of clustering in multilevel FM and the difficulty of multi-way partitioning are two fundamental gaps in knowledge). We seek such improvements while attempting to adhere to the research and reporting principles and standards that we have outlined in this paper.

References

[1] C. J. Alpert, "Partitioning Benchmarks for the VLSI CAD Community, `http://vlsicad.cs.ucla.edu/~cheese/benchmarks.html`

[2] C. J. Alpert, "The ISPD-98 Circuit Benchmark Suite", *Proc. ACM/IEEE International Symposium on Physical Design*, April 98, pp. 80-85. See errata at `http://vlsicad.cs.ucla.edu/~cheese/errata.html`

[3] C. J. Alpert, J.-H. Huang and A. B. Kahng,"Multilevel Circuit Partitioning", *ACM/IEEE Design Automation Conference*, pp. 530-533.

[4] C. J. Alpert and A. B. Kahng, "Recent Directions in Netlist Partitioning: A Survey", *Integration*, 19(1995) 1-81.

[5] R. S. Barr, B. L. Golden, J. P. Kelly, M. G. C. Resende and W. R. Stewart, "Designing and Reporting on Computational Experiments with Heuristic Methods", *technical report* (extended version of *J. Heuristics* paper), June 27, 1995.

[6] F. Brglez, "ACM/SIGDA Design Automation Benchmarks: Catalyst or Anathema?", *IEEE Design and Test*, 10(3) (1993), pp. 87-91.

[7] F. Brglez, "Design of Experiments to Evaluate CAD Algorithms: Which Improvements Are Due to Improved Heuristic and Which are Merely Due to Chance?", *technical report* CBL-04-Brglez, NCSU Collaborative Benchmarking Laboratory, April 1998.

[8] T. Bui, S. Chaudhuri, T. Leighton and M. Sipser, "Graph Bisection Algorithms with Good Average Behavior", *Combinatorica* 7(2), 1987, pp. 171-191.

[9] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Hypergraph Partitioning With Fixed Vertices", in *Proc. ACM/IEEE Design Automation Conf.*, June 1999.

[10] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Design and Implementation of the Fiduccia-Mattheyses Heuristic for VLSI Netlist Partitioning", *Proc. Workshop on Algorithm Engineering and Experimentation (ALENEX)*, Baltimore, Jan. 1999.

[11] P. K. Chan and M. D. F. Schlag and J. Y. Zien, "Spectral K-Way Ratio-Cut Partitioning and Clustering", *IEEE Transactions on Computer-Aided Design*, vol. 13 (8), pp. 1088-1096.

[12] J. Cong, H. P. Li, S. K. Lim, T. Shibuya and D. Xu, "Large Scale Circuit Partitioning with Loose/Stable Net Removal and Signal Flow Based Clustering", *Proc. IEEE International Conference on Computer-Aided Design*, 1997, pp. 441-446.

[13] W. Deng, *personal communication*, July 1998.

[14] 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

[15] S. Dutt and W. Deng, "VLSI Circuit Partitioning by Cluster-Removal Using Iterative Improvement Techniques", *Proc. IEEE International Conference on Computer-Aided Design*, 1996, pp. 194-200

[16] S. Dutt and H. Theny, "Partitioning Using Second-Order Information and Stochastic Gain Function", *Proc. IEEE/ACM International Symposium on Physical Design*, 1998, pp. 112-117

[17] C. M. Fiduccia and R. M. Mattheyses, "A Linear Time Heuristic for Improving Network Partitions", *Proc. ACM/IEEE Design Automation Conference*, 1982, pp. 175-181.

[18] M. R. Garey and D. S. Johnson, "Computers and Intractability, a Guide to the Theory of NP-completeness", W. H. Freeman and Company: New York, 1979, pp. 223

[19] I. P. Gent, S. A. Grant, E. MacIntyre, P. Prosser, P. Shaw, B. M. Smith and T. Walsh, "How Not To Do It", *research report* 97-27, Univ. of Leeds School of Computer Studies, May 1997.

[20] S. Hauck and G. Borriello, "An Evaluation of Bipartitioning Techniques", *IEEE Transactions on Computer-Aided Design* 16(8) (1997), pp. 849-866.

[21] L. W. Hagen, D. J. Huang and A. B. Kahng, "On Implementation Choices for Iterative Improvement Partitioning Methods", *Proc. European Design Automation Conference*, 1995, pp. 144-149.

[22] A. B. Kahng, "Futures for Partitioning in Physical design", *Proc. IEEE/ACM International Symposium on Physical Design*, April 1998, pp. 190-193.

[23] G. Karypis and V. Kumar, "Analysis of Multilevel Graph Partitioning", draft, 1995

[24] G. Karypis and V. Kumar, "Multilevel k-way Partitioning Scheme For Irregular Graphs", Technical Report 95-064, University of Minnesota, Computer Science Department.

[25] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel Hypergraph Partitioning: Applications in VLSI Design", *Proc. ACM/IEEE Design Automation Conference*, 1997, pp. 526-529. Additional publications and benchmark results for hMetis-1.5 are available at `http://www-users.cs.umn.edu/~karypis/metis/hmetis/main.html`

[26] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel Hypergraph Partitioning: Applications in VLSI Domain", *technical report*, University of Minnesota Computer Science Department, March 27, 1998.

[27] G. Karypis and V. Kumar, "Multilevel Algorithms for Multi-Constraint Graph Partitioning", Technical Report 98-019, University of Minnesota, Department of Computer Science.

[28] G. Karypis and V. Kumar, "hMetis: A Hypergraph Partitioning Package Version 1.5", *user manual*, June 23, 1998.

[29] B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", *Bell System Tech. Journal* 49 (1970), pp. 291-307.

[30] B. Krishnamurthy, "An Improved Min-cut Algorithm for Partitioning VLSI Networks", *IEEE Transactions on Computers*, vol. C-33, May 1984, pp. 438-446.

[31] L. T. Liu, M. T. Kuo, S. C. Huang and C. K. Cheng, "A Gradient Method on the Initial Partition of Fiduccia-Mattheyses Algorithm", *Proc. IEEE International Conference on Computer-Aided Design*, 1995, pp. 229-234.

[32] L. Sanchis, "Multiple-way network partitioning with different cost functions", *IEEE Transactions on Computers*, Dec. 1993, vol.42, (no.12):1500-4.

[33] G. R. Schreiber and O. C. Martin, "Procedure for Ranking Heuristics Applied to Graph Partitioning", *Proc. 2nd International Conference on Metaheuristics*, July 1997, pp. 1-19.

[34] G. R. Schreiber and O. C. Martin, "Cut Size Statistics of Graph Bisection Heuristics", *manuscript* in submission to *SIAM J. Optimization*, 1997.

[35] P. R. Suaris and G. Kedem, "Quadrisection: A New Approach to Standard Cell Layout", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 1987, pp. 474-477.

[36] W. Sun and C. Sechen, "Efficient and Effective Placements for Very Large Circuits", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 1993, pp. 170-177.

[37] Y. C. Wei and C. K. Cheng, "Towards Efficient Design by Ratio-cut Partitioning", *Proc. IEEE International Conference on Computer-Aided Design*, 1989, pp. 298-301.