Quantified Suboptimality of VLSI Layout Heuristics*

Lars W. Hagen†, Dennis J.-H. Huang and Andrew B. Kahng

UCLA Department of Computer Science, Los Angeles, CA 90024-1596
† Cadence Design Systems, Inc., San Jose, CA 95134

Abstract

We show how to *quantify* the suboptimality of heuristic algorithms for NP-hard problems arising in VLSI layout. Our approach is based on the notion of constructing new *scaled* instances from an initial problem instance. From the given problem instance, we essentially construct doubled, tripled, etc. instances which have optimum solution costs at most twice, three times, etc. that of the original instance. By executing the heuristic on these scaled instances, and then comparing the growth of solution cost with the growth of instance size, we can measure the *scaling suboptimality* of the heuristic. We give experimentally determined scaling behavior of several placement and partitioning heuristics; these results suggest that siginificant improvement remains possible over current state-of-the-art methods.

1 Introduction

For many problems in VLSI design, users will implicitly benefit from accurate estimates of the *suboptimality* of a given heuristic solution. Such estimates can be used to determine which heuristic should be used in a given application, or to determine the best allocation of design effort to various phases of layout. With cell placement and netlist partitioning in particular, the suboptimality of a heuristic directly affects the area and wire estimates used for performance estimation and high-level design space exploration. However, accurate estimates of suboptimality are difficult to obtain, for two reasons:

- First, determining the optimum solution is usually intractable, and theoretical bounds on the optimum solution cost may be quite loose. For example, spectral lower bounds for partitioning can be far from the optimum solution cost [3].
- Second, input constructions for which optimum solution costs are known (i.e., for which the error of a heuristic can be quantified) are often considered "artificial". For example, mesh- or chain-like topologies for placement, and the instances of Bui et al. [6], Garbers et al. [8] and Ackley [1] for partitioning, may be helpful in measuring algorithmic suboptimality but do not always give meaningful performance estimates for "real" examples.

In this paper, we propose a general measure of heuristic performance, based on the notion of the *scaling suboptimality* of a given heuristic. We believe that our work is interesting because it gives practically useful, *quantified* estimates of suboptimality for problem instances where there is no hope of knowing the optimal solution or establishing tight theoretical lower bounds on the optimal solution cost. From a given problem instance, our methodology essentially constructs 'doubled', 'tripled', etc. instances which have optimum solution costs at most twice, three times, etc. that of the original instance. Executing the heuristic on these *scaled instances*, then comparing the growth of solution cost with the growth of instance size, yields the scaling suboptimality of the heuristic.

Our approach can extend to provide estimates of optimal solution cost for specific problem instances, and also affords the means to construct a range of realistic test cases of arbitrary size. Beyond yielding insights into the relative utility of various heuristics as problem sizes grow large, our experimental results reinforce the need for continued research in layout design.

Previous estimates of heuristic suboptimality have centered on the construction of instances *for which the optimal solution cost is known*. As noted above, such methods have included the use of mesh and chain topologies for placement, and "difficult" (highly clustered or even disconnected) classes of partitioning instances [6, 8, 1]. Historically, the major objection to the constructive approach has been that the instances are artifactual and "not realistic". Two recent methods which start with "real" instances are thus of interest:

- Nakatake and Kajitani [11] generate a sequence of global routing instances, each of which has known minimum-possible maximum channel density. Then, [11] derives a parameter of the heuris-

*This research was supported in part by NSF grant MIP-9257982. ABK would like to thank Yoji Kajitani and Majid Sarrafzadeh for a discussion at Schloss Dagstuhl, October 1993.

tic, analogous to the parameter we develop below, which measures the growth rate of the maximum channel density.

- A construction of Boese [5] can be used to estimate the suboptimality of cell-based placement algorithms. Given a netlist hypergraph $G_H = (V, E)$ and an array of $|V|$ placement slots, the idea is to construct a new hypergraph G'_H which optimally assigns the terminals of each hyperedge onto a number of contiguous slots equal to the hyperedge size. The resulting G'_H can be "difficult" to distinguish from G_H, yet the optimum placement cost (Manhattan wirelength) of G'_H is known.

The rest of this paper is organized as follows. In Section 2, we define the scaling parameter of suboptimality. Sections 3 and 4 respectively describe the application of the scaling parameter, along with experimental results, for cell placement and netlist bipartitioning. Section 5 summarizes the implications of the results that we present.

2 The Scaling Parameter of Suboptimality

Consider a heuristic H for a given combinatorial (minimization) problem. For any problem instance I, we say that executing H on I yields a solution with cost $c_H(I)$, while the cost of the optimal solution is denoted $c^*(I)$.

Given any instance I with optimal solution cost $c^*(I)$, and any positive integer k, suppose that we can construct a new instance kI which has optimal solution cost at most $k \cdot c^*(I)$. It is easy to see that $k \cdot c_H(I) \geq k \cdot c^*(I) \geq c^*(kI)$. Thus, if executing H on the instance kI yields a solution with cost $c_H(kI) > k \cdot c_H(I)$, we have a *lower bound* for the suboptimality of heuristic H on instance kI.

In other words,

$$\frac{c_H(kI)}{c^*(kI)} \geq \frac{c_H(kI)}{k \cdot c^*(I)} \geq \frac{c_H(kI)}{k \cdot c_H(I)}$$

so that whenever $\frac{c_H(kI)}{k \cdot c_H(I)} > 1$, we immediately know that the heuristic is suboptimal (the error is at least $\frac{c_H(kI)}{k \cdot c_H(I)} - 1$). The key point is that we do not need to know the optimum solution costs $c^*(I)$ or $c^*(kI)$: we have "bootstrapped" an estimate of suboptimality.

The construction of instance kI from instance I allows us to experimentally determine the error $\alpha_H(k) = \frac{c_H(kI)}{k \cdot c_H(I)} - 1$ as a function of k (e.g., we might find that $\alpha_H(k)$ grows linearly or exponentially with k). We propose to use $\alpha_H(k)$ as a measure of the *scaling suboptimality* of the heuristic H. Several interesting applications immediately arise.

- The growth of $\alpha_H(k)$ can be used to predict the continuing utility of various heuristics as problem sizes increase. For example, we may find that a certain heuristic is most useful for a particular range of instance sizes.
- Similarly, the functions α_{H_1} and α_{H_2} corresponding to two heuristics H_1 and H_2 could be used to determine which heuristic is asymptotically better. For example, if there is some k_0 such that for all $k \geq k_0$, $\alpha_{H_1}(k) < \alpha_{H_2}(k)$, then we might say that heuristic H_1 dominates heuristic H_2.
- If we know the (expected value of the) function α_H for any given instance size n, we obtain a new estimate for the optimum solution cost of such an instance:[1] simply execute H and divide by $1 + \alpha_H(n)$.

Thus, the idea of scaling suboptimality can potentially lead to *quantified* lower error estimates for arbitrary optimization heuristics. For a given problem domain, the key question is whether we can find an appropriate construction of scaled instances kI from a given instance I. With such problems as the planar traveling salesman problem or graph coloring, some constructions appear more useful than others, but the best constructions are not clear. However, for several basic VLSI layout optimizations, we have devised very straightforward scaling constructions, as described in Sections 3 and 4 below. In the context of cell placement we demonstrate our methodology using two leading standard-cell layout tools (the TimberWolf7.0 place-and-route package [13] and the GORDIAN-L package [14]), and in the context of netlist bipartitioning we use Fiduccia-Mattheyses (FM) [7], two-phase matching-based clustering FM [6], the EIG1 spectral method [9], and the RCut1.0 ratio cut partitioner [17].

3 Scaling Suboptimality in Cell Placement

Our first set of experiments examines the scaling behavior of heuristics for cell placement. In order to construct scaled instances from a given real netlist, we simply replicate the netlist I so that the new instance kI consists of k disjoint copies of I; the optimum layout area for kI cannot be worse than k times the optimum layout area for I. While this construction may seem "unrealistic", note that each copy of I is presumably a "real" netlist. We believe that using disjoint copies of I makes matters simpler for the placement algorithm in that, e.g., the connected components of kI can be processed separately. Therefore, if we find a scaling suboptimality α_H using our construction of instances kI, we may interpret it as evidence of poor scaling by the heuristic.

For analytic placement methods that are based on quadratic optimization and that require a connected netlist and prescribed pad locations, we have an alternative construction which maintains pads and connectivity in the scaled instances. We accomplish this by removing all but four pads from the original netlist and locating one pad on each side of the layout region. In addition, we make sure that each pad is connected to only one cell by a single net. We can now construct

[1] Where no confusion can result, we overload the definition of α_H to encompass either (i) the suboptimality $\alpha_H(k)$ associated with scaling the instance by a factor k, or (ii) the suboptimality $\alpha_H(n)$ associated with a given absolute instance size n.

scaled copies of the benchmark using the construction shown in Figure 1. This construction will maintain connectivity,[2] and the directional "pull" from the pads is maintained by our method of connecting eliminated pads. In addition, it is easy to continue replicating any scaled instance, since it already satisfies our constraints with respect to the number of pads and their location. As in the case of the disjoint scaled construction, we believe that any suboptimality in α_H provides evidence of poor scaling by the heuristic.

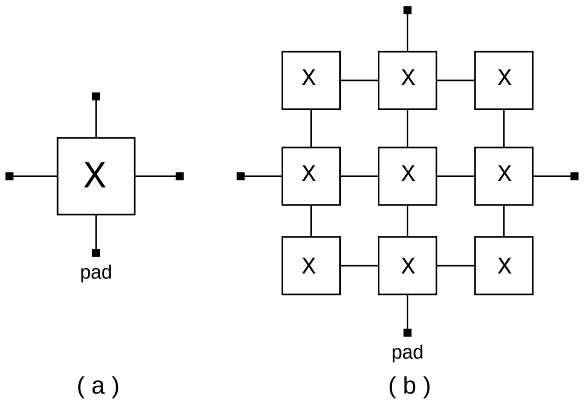

Figure 1: (a) Modified instance for single copy of benchmark X. Four modules are connected to the pads at the center of each side. (b) A scaled instance of benchmark X which consists of nine copies of benchmark X. The four pads of the "multiplied" benchmark are the pads belonging to the copies of benchmark X at the center of each side; remaining unconnected pads are eliminated.

Experimental Results

We have run experiments to test the scaling suboptimality of two "industrial-strength" placement and routing tools: the TimberWolf7.0 package [13, 15] and the GORDIAN-L package [14]. The basic optimization engine of TimberWolf7.0 is simulated annealing, and thus TimberWolf7.0 enjoys the attractions of the general simulated annealing approach – in particular, that the results (given sufficient CPU time and an appropriate annealing temperature schedule) are "asymptotically optimal".[3] GORDIAN-L is an analytic approach and arrives at a deterministic cell placement by performing quadratic optimization with an objective that iteratively approximates a linear wirelength objective.

We construct the scaled instances for our TimberWolf7.0 experiments using k disjoint copies of the original netlist, as outlined above. The Primary1 benchmark with all pads removed serves as the original instance I, and we invoke TimberWolf7.0 to compute a good placement of the remaining "core". We then construct a new instance kI consisting of k unconnected copies of the original core, and run TimberWolf7.0 on this multiple-copy scaled instance. To ensure that the final layout remains roughly square, we set the number of placement rows equal to $\sqrt{k}$ times the number of rows in the layout of the original instance. Given that the simulated annealing approach is – in theory – asymptotically optimal, and given the substantial CPU requirements of TimberWolf7.0 (the runtimes increase rapidly with k), we feel that our results are quite surprising. As shown in Table 1, the layout areas for the core multiples show that TimberWolf7.0 has over 10% area suboptimality for the largest netlist.

Circuit	#Runs	Area (10^7)	Range (10^7)	Ratio
Prim1	10	1.70	(1.68–1.71)	1.00
Prim1x4	10	6.90	(6.82–7.04)	4.06
Prim1x9	10	15.9	(15.6–16.3)	9.35
Prim1x16	10	29.1	(28.3–29.3)	17.12
Prim1x25	10	46.4	(45.8–46.7)	27.29
Prim1x36	6	68.6	(67.9–69.3)	40.35

Table 1: Layout area results for TimberWolf7.0 on multiple copies of the Primary1 benchmark netlist. Ratio is the *average* layout area for the multiple-copy (scaled) instance divided by the *average* layout area for the single-copy (original) instance.

Our GORDIAN-L experiments construct scaled versions kI by connecting k copies of the original netlist and four pads, as outlined above. Again, we use the Primary1 benchmark and remove all but four of the pads from the original design; we invoke GORDIAN-L to compute a good placement of this instance I. The scaled instances kI consisting of k connected copies of instance I are then constructed and passed to GORDIAN-L for placement. The performance measure used by GORDIAN-L is wirelength (measured as sum of bounding box half-perimeters and reported by GORDIAN-L itself) which, although not identical to area, is considered to be a reasonable placement objective. The results in Table 1 show that GORDIAN-L has notable scaling suboptimality. This is somewhat surprising in light of : (i) GORDIAN-L is a deterministic, "global" method, and one might expect the solution quality to scale better than that of an inherently "local" move-based algorithm such as TimberWolf7.0; and (ii) recent trends in CAD vendor tools show a migration to GORDIAN- or PROUD-style [16] approaches. Very recent work [12] has shown that GORDIAN-L shows almost "perfect" scaling on a set of scaled instances constructed using a slightly different but similar methodology to the one we used.[4]

[2] Connectivity is maintained given that the original netlist is connected, which holds for all test cases that we discuss.

[3] Note that many problem- and technology-specific heuristics have been added over the years to enhance the annealing approach. For example, invoking TimberWolf7.0 using default parameters, as in our experiments, will result in clustering optimizations being applied before placement is performed.

[4] The methodology of Riess [12] preserved nearly all of the original pads, thereby resulting in instances with greater pull

Although this result does not explain the poor scaling of GORDIAN-L shown in our experiments, it does raise the question of whether the construction we use is somehow biased against GORDIAN-L.

Circuit	Half-Perimeter (10^6)	Ratio
Prim1	2.14	1.00
Prim1x2	4.86	2.27
Prim1x4	10.18	4.76
Prim1x9	26.70	12.48
Prim1x16	48.76	22.78

Table 2: Wirelength (measured as sum of bounding box half-perimeters) results for GORDIAN-L on multiple copies of the Primary1 benchmark netlist. Ratio is the wirelength for the multiple-copy (scaled) instance divided by the wirelength for the single-copy (original) instance.

4 Scaling Suboptimality in Netlist Bipartitioning

Our second set of experiments examines the scaling behavior of netlist bipartitioning heuristics. For the bipartitioning problem, we can construct scaled instances kI by simply "tying" k copies of a given netlist I together such that the optimum bisection of kI must bisect each of the k copies of I. A construction which accomplishes this is now described. Given an instance I with n nodes and m edges, we can double I by constructing an instance $2I$ which contains both I and I', an isomorphic copy of I. Next, we add an edge to $2I$ between each node in I and the corresponding node in I', and assign some large weight M to each of these n edges. If we set $M = \infty$, the new instance $2I$ will trivially have optimal partitioning cost that is twice the optimal partitioning cost for instance I (i.e., $c^*(2I) = 2 \cdot c^*(I)$. However, setting $M = \infty$ will make it impossible for any heuristic to split the two copied nodes, i.e., the resulting doubled instance is essentially identical to a copy of I where the weight of each edge has been doubled.

To make scaled instances that have "scaling attributes", we propose setting $M = degree(v)$, where $degree(v)$ is the number of edges incident to node v in instance I. In other words, we create an edge (v, v') connecting $v \in I$ and its corresponding node $v' \in I'$, and assign it weight $w(v, v') = degree(v)$. The following theorem proves that our construction of $2I$ has the desired property $c^*(2I) = 2 \cdot c^*(I)$.

Theorem 1 : *If instance I has optimal bisection cutsize d, then $2I$ has optimal bisection cutsize $2d$.*

Proof : We can bisect $2I$ with cutsize $2d$ by simply bisecting each of the two isomorphic copies in the same place as the optimal bisection in I. This is shown as the $(A|\bar{A})$ cut in Figure 2(b), with cutsize $C(A, \bar{A}) = 2d$.

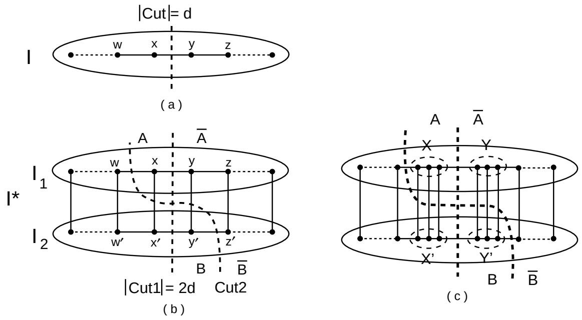

Figure 2: (a) The optimal cut for instance I. (b) Cut1 $= (A|\bar{A})$ is the optimal cut for instance $2I$. Cut2 $= (B, \bar{B})$ is the cut that results if nodes x and y' are swapped. (c) Result of swapping node sets X and Y'.

We first show that swapping any pair of nodes across the $(A|\bar{A})$ cut will yield a new cut with cutsize $\geq 2d$. If both the swapped nodes are in either I_1 or I_2, this is equivalent to swapping nodes x and y in Figure 2(b), which results in a new cutsize $\geq 2d + w(x, x') + w(y, y')$. Otherwise, without loss of generality we can assume that nodes $x \in A \cap I_1$ and $y' \in \bar{A} \cap I_2$ in Figure 2(b) are being swapped. This results in a cutsize

$$C(B, \bar{B}) = w(x, x') + C(B \cap I_1, \bar{B} \cap I_1) + w(y, y') + C(B \cap I_2, \bar{B} \cap I_2)$$

where $B = A \cup \{y'\} - \{x\}$. Since $C(B \cap I_1, \bar{B} \cap I_1) \geq C(B \cap I_1, \bar{A} \cap I_1)$ and $w(x, x') = degree(x)$, we have

$$\begin{aligned} w(x, x') &+ C(B \cap I_1, \bar{B} \cap I_1) \\ &\geq degree(x) + C(B \cap I_1, \bar{A} \cap I_1) \\ &\geq C(\{x\}, \bar{A} \cap I_1) + C(B \cap I_1, \bar{A} \cap I_1) \\ &= C(A \cap I_1, \bar{A} \cap I_1) \end{aligned}$$

Similarly, we can show $w(y, y') + C(B \cap I_2, \bar{B} \cap I_2) \geq C(A \cap I_2, \bar{A} \cap I_2)$. Thus,

$$C(B, \bar{B}) \geq C(A \cap I_1, \bar{A} \cap I_1) + C(A \cap I_2, \bar{A} \cap I_2) = C(A, \bar{A})$$

To prove the general result, observe that any bisection can be obtained by swapping m pairs of nodes from cut $(A, \bar{A})$, where $1 \leq m \leq n$. Assume, again without loss of generality, that we swap the set X of nodes in $A \cap I_1$ with the set Y' of nodes in $\bar{A} \cap I_2$ as shown in Figure 2(c). Using an argument similar to the case of a single pair of nodes, we have

$$\begin{aligned} &C(B, \bar{B}) \\ &= C(X, X') + C(B \cap I_1, \bar{B} \cap I_1) + C(Y, Y') + C(B \cap I_2, \bar{B} \cap I_2) \\ &\geq C(A \cap I_1, \bar{A} \cap I_1) + C(A \cap I_2, \bar{A} \cap I_2) \\ &= C(A, \bar{A}) \end{aligned}$$

□

towards the edges. It should be noted that since the TimberWolf7.0 cost measures total core area (cells + wiring), while GORDIAN-L cost measures wiring area alone, the scaling parameters are not directly comparable.

Theorem 1 shows that our construction will result in partitioning instances that are well-suited for testing the scaling of bisection heuristics. We also test the scaling suboptimality of two algorithms which address the ratio cut bipartitioning objective of Wei and Cheng [17]. The ratio cut objective minimizes $\frac{C(U,W)}{|U|\cdot|W|}$ where $C(U,W)$ is the number of signal nets crossing between the two partitions U and W. When a ratio cut instance is doubled as described above, the new instance $2I$ will have optimal solution cost $c^*(2I) = \frac{2\cdot C(U,W)}{2|U|\cdot 2|W|} = \frac{1}{2}\cdot c^*(I)$ as long as the optimal cut does not split duplicated nodes. We believe that a result analogous to Theorem 1 holds for optimal ratio cut bipartitions when the same scaling construction is applied (i.e., the optimal ratio cut cost decreases by a factor of exactly two each time the instance is "doubled").

Experimental Results

We have run experiments to test the scaling suboptimality of four partitioning heuristics: (1) the FM algorithm [7] for bisection, (2) a two-phase Matching-Based Compaction and FM (MBC+FM) algorithm due to Bui et al. [6] that is also for bisection, (3) the EIG1 spectral method from [9] for ratio cut bipartitioning, and (4) the RCut1.0 [17] ratio cut partitioner. FM, MBC+FM, and RCut1.0 perform greedy local search from a given (random or constructed) starting solution, while EIG1 is a deterministic algorithm that relies on a "global" eigenvector computation.

Our FM and MBC+FM experiments compared the performance on a real benchmark instance with the performance on a "doubled" and a "quadrupled" instance of the same benchmark. The MBC+FM implementation we used performed a single clustering pass over the set of nodes by finding a maximum matching and then first ran FM on the clustered instance to get a "good" starting point for the second FM run on the original "flat" instance. Both the FM and MBC+FM results are based on 50 independent runs from random starting points. From the results in Table 3 it is clear that both FM and MBC+FM scale quite poorly: the heuristic cutsize seems to grow roughly quadratically (1, 4, 16, ...) while the optimal cutsize grows linearly (1, 2, 4, ...). It is particularly interesting to note that although the MBC+FM results are considerably better than the FM results, the MBC+FM scaling parameter is as bad as and sometimes worse than the FM scaling parameter.

Experiments with EIG1 and RCut1.0 compared ratio cut results for the real benchmark instance with ratio cut results for the "doubled" instance of the same benchmark. The results in Table 4 show a striking difference in that the EIG1 results scale nearly perfectly (i.e., Ratio = 0.5) while the RCut1.0 results do not – the ratio cut for the "doubled" instance is in many cases larger than the ratio cut for the original instance.

Test Circuit		FM		MBC+FM	
		Avg	Ratio	Avg	Ratio
Prim1	x 1	86	1.00	78	1.00
	x 2	328	3.81	272	3.49
	x 4	1079	12.54	866	11.10
Prim2	x 1	295	1.00	220	1.00
	x 2	1613	5.46	1008	4.58
	x 4	4260	14.44	3713	16.88
Test02	x 1	182	1.00	149	1.00
	x 2	677	3.71	650	4.36
	x 4	2457	13.50	1911	12.83
Test03	x 1	126	1.00	90	1.00
	x 2	615	4.88	521	5.79
	x 4	2415	19.16	1880	20.89
Test04	x 1	149	1.00	116	1.00
	x 2	639	4.28	589	5.08
	x 4	2556	17.15	1966	16.95
Test05	x 1	196	1.00	166	1.00
	x 2	976	4.97	913	5.50
	x 4	3634	18.54	3073	18.51
Test06	x 1	95	1.00	95	1.00
	x 2	637	6.70	571	6.01
	x 4	2306	24.27	1869	19.67

Table 3: Mincut bisection results for FM and MBC+FM heuristics on single, doubled, and quadrupled instances of standard benchmarks. The numbers in the table give the average number of cut edges for 50 runs. Ratios are the average cutsizes for the doubled and quadrupled instances divided by the average cutsize for the single (original) instance. All module areas were set to 1.0 and each partition was required to be within ±1.0 of half the total module area.

Test Circuit	EIG1			RCut1.0		
	x 1	x 2	Ratio	x 1	x 2	Ratio
Primary1	14.64	7.32	0.50	14.8	14.8	1.00
Primary2	4.70	2.29	0.49	5.6	6.4	1.14
Test02	19.43	10.06	0.52	10.0	13.2	1.32
Test03	9.45	4.275	0.45	13.9	16.1	1.16
Test04	5.85	2.93	0.50	11.1	22.0	1.98
Test05	6.13	3.06	0.50	5.7	7.1	1.25
Test06	12.75	6.385	0.50	11.5	11.4	0.99

Table 4: Ratio cut results for EIG1 and RCut1.0 on single and doubled instances of standard test cases. The values in the table are multiples of 10^{-5}. The RCut1.0 values are the best of 10 runs with different random seeds. All module areas were set equal to 1.0.

5 Conclusions

In conclusion, we have investigated the general problem of obtaining quantified estimates of suboptimality for heuristic algorithms in VLSI layout. Most previous works which estimate heuristic suboptimality rely on specific constructions for which optimal solution costs can be determined. In contrast, our approach relies on the simple notion of constructing doubled, tripled, etc. instances from a known "realistic" instance such that the doubled, tripled, etc. instances have optimum solution cost equal to *at most* twice, three times, etc. that of the original instance. By upper-bounding the optimal solution cost of the scaled instances, we can obtain lower bounds on the heuristic error. We have presented simple scaling constructions for the CAD domains of placement and partitioning, and run experiments to measure the scaling behavior of the current state-of-the-art algorithms.

Our results from Section 3 indicate that both TimberWolf7.0 and GORDIAN-L can be expected to arrive at solutions which may be far from optimal. This leads to some interesting speculations. In particular, we believe that our results indicate there is definite room for improvement over current placement algorithms (TimberWolf7.0 and GORDIAN-L are considered the current state of the art). Similarly, the results of Section 4 indicate that iterative partitioning methods, even in combination with clustering approaches, scale quite poorly. While spectral methods seem to scale better, they suffer from other limitations (memory requirements, inability to model variable areas, pre-placements, and other constraints, etc.). Again, there seems to be room for improvement over current partitioning methodologies.

A second observation is that the α parameter can lead to new layout area estimation methodologies. By quantifying the performance degradation of a given heuristic, it becomes possible to achieve an area estimate that is not only design-dependent, but also *algorithm*-dependent. Many existing estimation tools, such as Rent-based methods [10], give estimates that are "intrinsic" to the netlist topology (e.g., estimates of the minimum area needed to embed the design). On the other hand, for many applications the designer must know the area requirements of the placement that will result after *particular* design tools are applied.

For an α parameter-based estimation methodology to succeed, two assumptions about the netlist must be made: (1) the connection structure must be relatively homogeneous; and/or (2) we must be able to extract representative subgraphs of the design for the initial analysis. The first assumption may be reasonable in light of the local hierarchy and regularity that is typical in most VLSI designs. The second assumption may also be reasonable in light of vertex-ordering or graph-search techniques (e.g., [2]) which can be used to extract a closely-connected subgraph from the netlist. Such techniques can be applied from various locations in the netlist in order to obtain a sampling of subgraphs which can together serve as the original instance I in the scaling calculation.

References

[1] D. H. Ackley, *A Connectionist Machine for Genetic Hillclimbing*, Kluwer, 1987.

[2] C. J. Alpert and A. B. Kahng, "A General Framework for Vertex Orderings, With Applications to Netlist Clustering", *Proc. IEEE Intl. Conf. Computer-Aided Design*, 1994, to appear.

[3] C. J. Alpert and A. B. Kahng, "Recent Directions in Netlist Partitioning: A Survey," to appear in *Integration: the VLSI Journal*, 1995.

[4] E. B. Baum, "Iterated Descent: A Better Algorithm for Local Search in Combinatorial Optimization Problems", *Proc. Neural Information Processing Systems*, D. Touretzky, ed., 1987.

[5] K. D. Boese, *personal communication*, January 1994.

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

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

[8] J. Garbers, H. J. Promel, and A. Steger, "Finding Clusters in VLSI Circuits", *Proc. IEEE Intl. Conf. on Computer-Aided Design*, 1990, pp. 520–523. Extended version.

[9] L. Hagen and A. B. Kahng, "New Spectral Methods for Ratio Cut Partitioning and Clustering", *IEEE Trans. on CAD* 11(9), Sept. 1992, pp. 1074-1085.

[10] L. Hagen, A. B. Kahng, F. J. Kurdahi, and C. Ramachandran. "On the Intrinsic Rent Parameter and Spectra-Based Partitioning Methodologies." *IEEE Trans. Computer-Aided Design*, **13**(1):27–37, 1994.

[11] S. Nakatake and Y. Kajitani, "Channel-Driven Global Routing with Consistent Placement", *Proc. IEEE Intl. Conf. on Computer-Aided Design*, 1994, pp. 350-355.

[12] B. M. Riess, *personal communication*, January 1995.

[13] C. Sechen, *Placement and Global Routing of Integrated Circuits Using Simulated Annealing*, PhD thesis, Univ. of California, Berkeley, 1986.

[14] G. Sigl, K. Doll, and F. M. Johannes, "Analytical Placement: A Linear or a Quadratic Objective Function?", *Proc. ACM/IEEE Design Automation Conf.*, June 1991, pp. 427–432.

[15] W. J. Sun, K. Roy and C. Sechen, "Fast, High-Quality Placement for Large Circuits", *Proc. 4th ACM/SIGDA Physical Design Workshop*, Lake Arrowhead, April 1993, pp. 11-12.

[16] R.-T. Tsay, E. S. Kuh. and C.-P. Hsu, "PROUD: A Fast Sea-of-Gates Placement Algorithm", *In Proc. ACM/IEEE Design Automation Conf.*, 1988, pp. 318-323.

[17] Y. C. Wei and C. K. Cheng, "Towards Efficient Hierarchical Designs by Ratio Cut Partitioning", *Proc. IEEE Intl. Conf. on Computer-Aided Design*, 1989, pp. 298-301.