Benchmarking For Large-scale Placement and Beyond[*]

Saurabh N. Adya[†], Mehmet C. Yildiz[‡], Igor L. Markov[†],

Paul G. Villarrubia[♯], Phiroze N. Parakh[♭] and Patrick H. Madden[‡]

†The University of Michigan, Department of EECS, Ann Arbor, MI 48109-2122

‡SUNY Binghamton, Computer Science Department, Binghamton, NY 13902

♯ IBM Corp., 11400 Burnet Road, Austin, TX 78758

♭ Sierra Design Automation, Inc., CA[†]

November 6, 2003

Abstract

Over the last five years the VLSI Placement community achieved great strides in the understanding of placement problems, developed new high-performance algorithms, and achieved impressive empirical results. These advances have been supported by non-trivial benchmarking infrastructure, and future achievements are set to draw on benchmarking as well. In this paper we review motivations for benchmarking, especially for commercial EDA, analyze available benchmarks, and point out major pitfalls in benchmarking. Our empirical data offers perhaps the first comprehensive evaluation of several leading large-scale placers on multiple benchmark families. We outline major outstanding problems and discuss the future of placement benchmarking. Furthermore, we attempt to extrapolate our experience to circuit layout tasks beyond placement.

1 Introduction

Progress in VLSI placement research over the last five years has been tremendous. High-performance free placement tools such as `KraftWerk` [28], `Capo` [13], `Dragon` [61] and `Feng Shui` [65] are now widely available [16] and used. They have been successfully tested on ever-increasing circuits and, according to comparisons made at Intel, IBM and Cadence, are on par with industrial tools in terms of minimizing the *half-perimeter wirelength* (HPWL) and other simple placement objectives. Independent implementations of the `KraftWerk` algorithm [28] are used in industrial placers in at least three publicly-traded companies, including a major VLSI design house, a major FPGA vendor and a major EDA tool vendor. The `Capo` program [13] is being commercialized by at least two EDA start-ups and is also a part of the Amplify-ASIC tool from

[*]Contact author: Prof. Igor Markov `imarkov@umich.edu`.

[†]Dr. Parakh was with Monterey Design Systems when this work was conducted.

Synplicity, Inc. To this end, optimizing simple objectives well can be viewed as a necessary, but not sufficient requirement for a successful placement algorithm.

More importantly, we now have a much better understanding of such important issues as *a priori* interconnect prediction [58], routing congestion [53, 62] and timing [36, 63, 41]. Given that VLSI placement is largely an empirical field, much of this progress would have been impossible without the public availability of large circuit benchmarks [16], such as the 18 ISPD-98 circuits released by IBM [7] and their derivatives [61, 63]. The availability of open-source placers and public placement benchmarks leads to new synergies by allowing researchers to modify the tools and the benchmarks, analyze tool performance in depth and combine tools to solve new design problems [1]. Many open problems in placement remain, and benchmarking issues, such as independent replication of reported results, are integral to further progress.

The ability to accurately measure the impact of any technique is crucial to scientific advancement. Recently, the physics community had a surprising revelation: a number of papers from a well known researcher were based on fraudulent data [9, 43]. Apparently driven by the desire to publish, experimental results were fabricated, with theory being passed off as fact. Had it not been for the failure of other research groups to replicate experiments, the fraud might have gone undetected. Incorrect published results are damaging to subsequent research; while there is no indication in the Physical Design community of *intentional* misrepresentation of results, reported results often cannot be reproduced [46]. Even the most basic metrics are freely interpreted by different authors, and it is nearly impossible to determine the best approach for a given problem from published results. Research in algorithm development has additional pitfalls. In some cases, algorithms with attractive asymptotic worst-case complexity are not useful in practice, or vice versa. In other cases, proposed algorithms are much more complicated than existing algorithms and offer only marginal improvement. Therefore, algorithm designers must make an effort to show the utility of their proposed algorithms. To this end, open benchmarking allows one to reuse efforts of many researchers to perform comprehensive testing.

As circuits become larger and more complex, the need to improve design automation tools becomes more urgent. According to recent literature, existing tools may produce layouts several times worse than what is achievable. This has been independently shown (i) by comparing manually designed circuits to those laid out by commercial tools [25], and (ii) by placing specially designed circuits with known good layouts [31, 18]. Thus, much room for improvement remains in circuit layout, and even the slowdown of Moore's law would emphasize possible improvements due to EDA tools. Yet, if we cannot reliably compare EDA tools and relevant research results, significant progress is unlikely.

This paper reviews basic motivations for placement benchmarking in Section 2. Major available tools and benchmarks are analyzed in Section 3. Routability is discussed in Section 4 and timing in Section 5. We attempt to extrapolate the lessons learned to layout tasks beyond placement in Section 6.

2 Layout Sophistication Motivates Open Benchmarking for EDA

As VLSI chips grow in size and complexity, large-scale placement is becoming integral to achieving multiple design objectives. Some of the most important goals are the minimization of wirelength, routing congestion, cycle-time and power dissipation. These objectives may correlate, e.g., wirelength and power, but in some cases conflict with each other. For

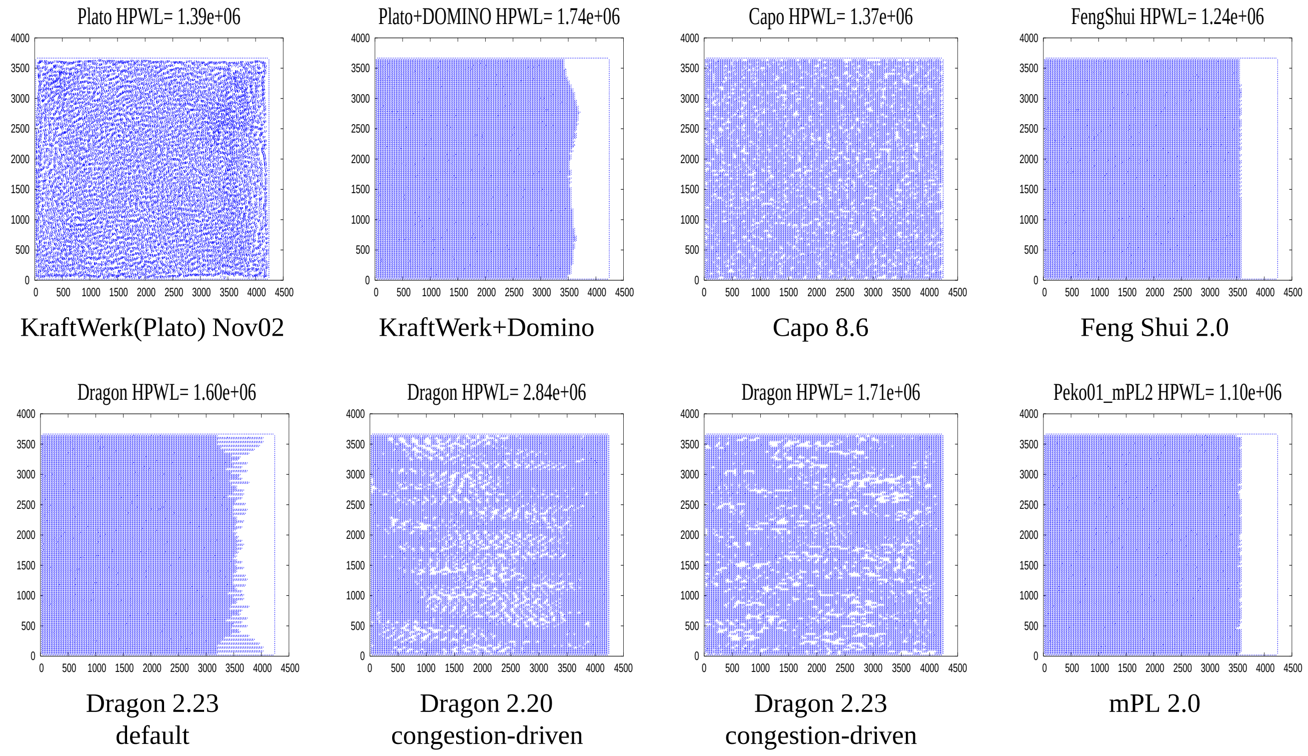


Figure 1: **Placements of the 12.5K-cell benchmark PEKO01 produced by placers `KraftWerk`, `Capo`, `Dragon`, `FengShui`, and `mPL`. The `KraftWerk` placement has many cell overlaps and must be further legalized with `Domino`, which costs 20% wirelength. `Capo` and `KraftWerk` tend to distribute whitespace uniformly across the fixed die to generically improve routability, but variable-die placers pack cells in rows to the left to minimize wirelength. In congestion-driven mode, `Dragon` behaves like a fixed-die placer and allocates whitespace according to an internal congestion map. Wirelengths in Table 2 may differ slightly for randomized placers such as `Capo` and `Dragon`.**

example, to avoid routing congestion in certain areas of ASIC designs, one may need to spread out collections of cells. This reduces wiring density, but increases wirelength.[1] Similarly, techniques that minimize timing in many cases increase wirelength and congestion. While experiences with specific optimization techniques provide only circumstantial evidence of conflicting objectives, for practical purposes this evidence is strong.

The multiplicity of conflicting objectives makes large-scale VLSI placement extremely complex. Additional complexity is due to design constraints, e.g., signal integrity guidelines, chip die and floorplan constraints, pre-designed on-chip intellectual property, etc.

Modern ASIC designs are laid out in the *fixed-die context*, where the layout area, routing tracks and power lines are fixed before placement starts [13]. Typical minimization objectives are congestion and timing.[2] Fixed-die layout is relevant for processes with over-the-cell routing on three or more metal layers and often applied to design blocks rather than whole chips.

Placement density is a new concern implied by the fixed-die context. We define placement density *in a region* as the ratio of (i) total area of movable cells in that region to (ii) the area available for placement of movable cells in the region. Another related term is *whitespace*%= 100% – *density*%. Similarly, the *average density*, also known as *row utilization*, is

[1]In principle, there may be congestion minimization techniques that do not increase wirelength. These should be considered too.

[2]Fixed-die placement is a departure from textbook physical design where routing tracks and chip area are determined during placement and routing. As such, it invalidates older benchmarks because area is not minimized anymore.

Circuit	Optimal WL *e*6	KraftWerk WL *e*6	+Domino WL *e*6	Capo 8.6 WL *e*6	Dragon 2.20 WL *e*6	FengShui 2.0 WL *e*6	mPL 2.0 WL *e*6
Peko01	0.81	1.39	1.74	1.29	1.46	1.24	**1.10**
Peko02	1.26	1.98	2.61	2.03	2.43	2.20	**1.76**
Peko03	1.50	3.02	3.78	2.66	2.93	2.77	**2.05**
Peko04	1.75	3.25	4.25	3.12	3.87	2.84	**2.31**
Peko05	1.91	3.92	4.79	3.16	3.79	3.03	**2.57**
Peko06	2.06	4.07	5.38	3.57	4.35	3.28	**2.78**
Peko07	2.88	5.73	7.56	5.07	6.24	4.82	**3.95**
Peko08	3.14	5.87	8.17	5.57	6.79	5.05	**4.99**
Peko09	3.64	8.52	10.74	6.47	7.72	5.80	**4.76**
Peko10	3.73	8.9	12.03	8.0	8.49	8.10	**6.59**
Avg	2.26	4.66	6.10	4.09	4.80	3.91	**3.28**

Table 1: **HPWL on 10 out of 18 PEKO benchmarks in suite 1.** `KraftWerk` **placements are legalized by** `Domino`**. Best results are shown in bold.** `Kraftwerk` **and** `Capo8.6` **are fixed-die placers.** `FengShui 2.0`**,** `mPL 2.0` **and detail placer** `Domino` **are variable-die placers and pack cells to the left of the die area.** `Dragon 2.20` **is also run in the variable-die mode.**

defined as the ratio of the total area of *all* movable cells to the total amount of area available for placement of movable cells. It cannot be changed by a fixed-die placer. If all movable cells are uniformly spread throughout the layout area, the average density will be achieved at all locations. To improve yield, placers may be required to limit density in any given region (uniform distribution of whitespace is one of many ways to satisfy such a constraint). Observe that maximum density values (per region) below the average density are not feasible, and if the maximum density per region equals the average density, then whitespace must be distributed equally. However, placers are often allowed to allocate significant "free space" rather than distribute cells uniformly. Depending on the type of design, this may be an important part of the overall problem. Figure 1 shows how several academic placers handle 15% whitespace in the PEKO01 benchmark [18]. Observe that `Feng Shui`, `mPL` and the default version of `Dragon` pack cells to the left, leaving large empty areas on the right. Compared to the fixed-die (`-fd`) version of `Dragon` that distributes whitespace to alleviate congestion, the default version improves wirelength by around 6% for 15% of whitespace (by more when more whitespace is available). However, this significantly worsens routability [62]. Because of this, listing wirelengths achieved by fixed-die and variable-die placers, in the same table, may create a false impression of a two-sided comparison. To this end, if a variable-die placer produces similar or higher wirelength, then it is probably not a very good placer. Otherwise, one has to compare relevant design objectives, such as congestion.

Industrial placement instances, e.g., at IBM, can be classified into ASICs, SOCs, and Microprocessor RLMs (random logic macro). Each of these categories presents unique difficulties. ASIC chips generally contain a large number of fixed I/O ports that may be perimeter-restricted or pervasive throughout the core area of the chip (area-array I/O). ASIC chips frequently contain a handful (1-20) of pre-placed large macros that are fixed, a moderate number (100s) of movable large multi-row cells, and many small movable cells — up to several million and increasing. ASIC chips come in a variety of average densities typically ranging from 40% to 80%.

SOC designs are similar to ASIC designs, but with many more large macros fixed in the placement area. In extreme cases the bulk of the design is concentrated in standard pre-designed library cores, RAMs, etc, with only a small fraction

Circuit	#Nodes	#Nets	WS %	Optimal HPWL	Dragon2.23 HPWL	Plato HPWL	FengShui2.0 HPWL	mPL2.0 HPWL	mPL2.0 % overlap	Capo8.5 HPWL	Capo8.6 HPWL
10x10	100	184	0	184	293	202	247	168	22	267	**184**
95x95	9025	17864	5	17884	39687	**18302**	31304	37735	1.89	21828	22764
100x100	10000	19804	0	19804	46066	**20519**	33397	40219	2.02	38352	21314
190x190	36100	71824	5	71864	175623	**75384**	155779	151134	0.94	90665	89814
200x200	40000	79604	0	79604	198182	**82335**	137841	160700	1.00	193167	100041
Avg				37856	91970	**39348**	71713	77991	-	68855	46823

Table 2: **Wirelength achieved by several placers on regular grids of varying size and with varying whitespace. `Plato` is the original implementation of `KraftWerk` by Eisenmann and Johannes. While `Plato` produces small wirelength on $n \times n$ grids, it often fails to converge to a solution on random-logic netlists with embedded grids. `FengShui 2.0`, `mPL 2.0` and `Capo 8.5` frequently produce suboptimal solutions. However, `Capo 8.6` does reasonably well. Since `mPL 2.0` frequently produces overlaps, we also report the overlap as a % of the layout area.**

of movable logic providing minor control functions. Such placement instances tend to have extremely low densities on the order of 20%, and in some cases less than 5%. Placement algorithms developed for nearly-full designs often do not handle such extremes well. Therefore one seeks algorithms specifically developed for this context and tested on low-density designs [4].

Microprocessor designs are generally laid out hierarchically, and this approach often leads to many small partitions. Some of these partitions are small standard-cell placement instances with very few fixed cells, a large number of fixed I/O ports, and a small number of movable cells ($<$ 10000). Because densities tend to be high, averaging 80% and reaching 98%, specialized techniques are needed to produce good placements that are also legal [64, 15]. Also, due to the small number of movable objects and the large number of fixed ports [8], multi-level partitioning [6, 37] is no better during placement than simpler “flat” FM partitioning [29]. Related results have been reported in [11, 8].

Given the complexity and the variety of VLSI placement problems, one should not expect a single closed-form algorithm or even a commercial tool to work well in all circumstances. Comprehensive evaluation of algorithms and tools is non-trivial, and so is testing the applicability to each of the relevant domains. Often, algorithms that perform well in the case of an RLM do not perform well for an ASIC (e.g., recursive bisection with “flat” FM partitioning). Likewise, algorithms such as recursive bisection with multi-level partitioning that show significant promise on very large netlists may provide little or no value for small structured components of a microprocessor chip. Some algorithms may perform well on dense designs while others perform well on sparse designs. Other differentiating factors include the diversity of cell sizes and the presence of fixed and movable macros, — these are increasingly important in modern designs.

The sophistication and variety of layout problems, as well as the multitude of performance factors, make the case for public benchmarking in Physical Design. Indeed, we observe a similar situation in computer architecture, where different design decisions may favor different applications, and the variety of microprocessors feeds the need for comprehensive comparisons. To this end, industry-standard evaluation of desktop and server hardware performance across a variety of tasks is based on SPECmark benchmarks [57]. Originally, the Standard Performance Evaluation Corporation (SPEC) published a suite of 10 benchmarks that test a computer’s integer and floating point computation. The suite includes slightly hacked versions of well-known FORTRAN and C codes. The performance measure of one SPECmark is comparable to that of a VAX-11/780. Additional SPECmark suites have been published in recent years.

Similarly, an appropriate set of placement benchmarks could become a standard for measuring and categorizing the behavior of placement algorithms. Understanding how algorithms behave across the entire problem space is important for

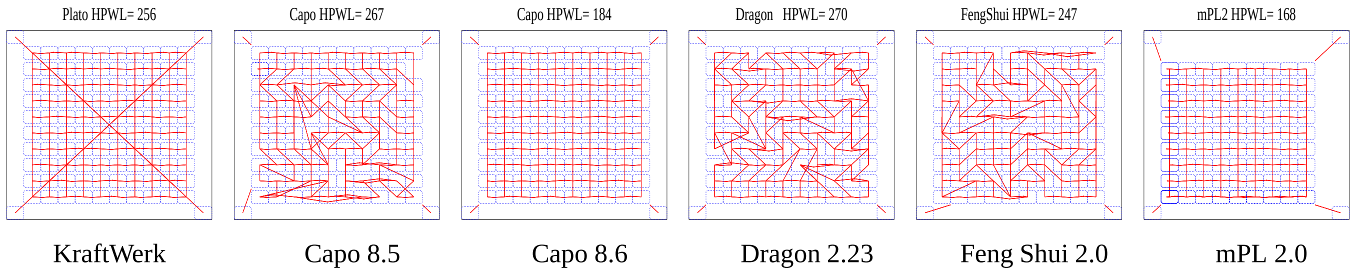


Figure 2: **Placements of a 10x10 grid-graph produced by academic placers. All pictures are produced using a utility that ships with `Capo` and gnuplot. `mPL 2.0` places cells with many overlaps, and `Capo 8.5` produces one overlap. `mPL 1.2b` (not shown) and `Capo 8.6` both find the unique optimal placement. `KraftWerk` produces good placements, but the block of movable cells has a wrong orientation relative to fixed pads. For larger grids, none of the placers produce optimal wirelength. The smallest wirelength is achieved by `KraftWerk`, then `Capo 8.6`, closely followed by `mPL 2.0`. All three are generally within 25% from optimal HPWL. `Dragon 2.20` and `Capo 8.5` double wirelength.**

selecting and developing the best techniques, and such an effort seems beyond any single industrial or academic placement group. The industry needs placement benchmarking to improve internal tool development, to measure potential vendor tool offerings, and to communicate important issue to academic researchers. Such benchmarks can greatly enhance the efficiency of communication between all parties involved, especially when studying new design features.

Some design features may depend on technology generations, therefore we would not like to confine placement benchmarking to a rigid testing protocol. In fact, we hope that the detailed placer comparisons on existing benchmarks in this paper will encourage researchers to work on new, more up-to-date layout benchmarks and optimization objectives. Indeed, besides total wirelength and congestion, layout tools must optimize timing and ensure signal integrity. Such design objectives and constraints must be reflected in public placement benchmarks so that one can compare layout tools in a variety of placement contexts. In particular, one would like to see trustworthy empirical data for academic placers that lend themselves to timing closure flows and produce good timing results while maintaining routability and signal integrity. Such community benchmarks are relevant for users of commercial EDA tools, and an investment into producing them is justified in the long term. An attempt at such benchmarks is being made at CMU [55].

3 Available Open Benchmarks and Placement Tools

As noted in [46], published results for wirelength-driven placement of MCNC benchmarks differ wildly due to creative but poorly explained interpretation of input files by authors. Similarly, we show in Section 5 that more recent published results on timing-driven placement exhibit alarming contradictions. Less drastic differences may go unnoticed, or even be falsely advertised as algorithmic improvements, especially when reported implementations are not available for independent evaluation. For example, some works on placement and floorplanning claim good wirelength, but their placements have many cell overlaps. Another source of discrepancies in published results is poorly specified benchmarks that require pre-processing and additional information, e.g., timing-driven placement benchmarks from [22] published only in Verilog. Limited parsers also cause difficulties, e.g., as of February 2003, the second suite of IBM-Dragon benchmarks in LEF/DEF has two variants — one for `Dragon` and one for `Capo`. To this end, we advocate using common, recent benchmarks and standard parsers,

Feature	KraftWerk	Capo 8.6	Dragon 2.23	FengShui 2.0	mPL 2.0
Fixed Terminals	+	+	+	+	+
Macros	±	±	-	-	-
Terminals of arbitrary size	+	+	-	+	±
Cells of arbitrary size	+	+	+	+	±
Subrows	-	+	±	±	-
Orientation constraints	-	+	-	-	-
LEF/DEF	-	+	±	±	-
Whitespace Management	±	±	±	-	-
Detail Placement	-	±	+	±	±
Timing-driven Placement	+	±	+	-	-

Table 3: **Features supported by different academic placers. We categorize support for various features as supported(+), absent(-) and partially supported(±). `KraftWerk` and `Capo` support fixed macros, but often fail to remove overlaps of movable macros [1]. Being a global placer `Capo` comes with a very simple detail placer. `CapoT`, a timing-driven version of `Capo` is not open-source at the moment.**

Circuit	#Nodes	#Nets	KraftWerk Time(min)	+Domino Time(min)	Capo8.6 Time(min)	Dragon2.23 Time(min)	FengShui1.1 Time(min)	FengShui2.0 Time(min)	mPL1.2b Time(min)	mPL2.0 Time(min)
ibm10	67692	64227	22	a	11	99	15	164	14	74
ibm17	183102	180684	104	a	55	360	79	729	53	607
ibm18	210323	200565	73	a	50	340	74	a	121	818

Table 4: **Time taken by various placers on the three IBM Dragon v1 benchmarks. `mPL1.2b` and `mPL 2.0` are run on a 750 MHz Sun Blade-1000 workstation running Solaris. All other placers are run on 2 GHz Pentium/Linux system. The Pentium/Linux systems are generally 2X faster than the Sun Blade-1000 workstation. `mPL2.0` and `Dragon` take long times placing these netlists. `FengShui 2.0` is also very slow, and the bottleneck is apparently in detailed placement.**

e.g., Cadence's open-source LEF/DEF parsers downloadable for free from `http://www.openeda.org`. Appendix A surveys existing families of placement benchmarks, and Appendix B describes downloadable placers. We typically cite the publication where a given contribution was first described, but download links for software and circuits can be found in the GSRC Bookshelf [16], specifically in the Wirelength-Driven Placement and Circuit Design Examples slots. Based on empirical data for various pairings of benchmarks and placers, we observe interesting trends.

Table 4 reports average placer runtimes. `Dragon`, `mPL2.0` and `Feng Shui 2.0` appear to be the slowest placers. Note that since some of the placers are randomized, running them several times and taking the best solution out of N allows one to trade off runtime for improved solution quality. Figure 3 illustrates this trade-off for `Capo`. It shows that the second independent start improves the HPWL by one percent on average, and additional starts are typically not worth the runtime.

Table 3 is a check-list for common features found in placers. `Dragon` and `Capo` tend to support more features than other placers. We categorize support for various features as supported(+), absent(-) and partially supported(±). Some of the features we took into consideration are discussed below.

- **Macros**: `Kraftwerk` is claimed to be a generic placement tool, capable of handling large macros during the placement process. However, for constrained designs it often produces layouts with large amounts of overlap. `Capo` only supports fixed macros. Other placers do not have any support for macros.

- **Subrows**: Subrows are often created in the layout because of obstacles like pre-placed macros, power strips etc. `Capo` has support for subrows. However, most of other placers do not have adequate support for subrows. `Kraftwerk` does not attempt to put the standard-cells into rows. It uses the detail placer `Domino` to pack cells into rows.
- **Orientation Constraints**: Standard-cell rows often have constraints on the orientation of the cells placed in them.
- **LEFDEF**: `Capo` contains two pairs of LEF/DEF parsers, of which one is the code released by Cadence in public domain. For any particular input file, either parser can be selected. `Dragon` and `Feng Shui` support fairly limited subsets of LEF/DEF.
- **Whitespace Management**: `Capo` and `KraftWerk` tend to distribute cells uniformly [15]. `Dragon`, `Feng Shui` and `mPL` in default modes pack cells to the left. `Dragon` in the fixed-die mode distributes whitespace to improve congestion [62].
- **Detail Placement**: `Capo` and `Feng Shui` implement naive detailed placers based on optimal re-placement of small groups of adjacent cells [14, 34]. `mPL2.0` also replaces small groups of cells, but those cells do not have to be adjacent. `Dragon` contains a detailed placer based on simulated annealing.

Empirical Analyses. Figure 1 plots the outputs of six placers on the PEKO01 benchmark [18] and suggests that `Dragon` in congestion-driven mode, `KraftWerk` and `Capo` behave like fixed-die placers. The other three simply pack cells in rows to the left, which is typical of variable-die placers. Even when this produces good wirelength (e.g., in `mPL`), such placements may be unroutable. On the other hand, `Dragon`'s congestion-driven mode doubles wirelength for PEKO01 and seems wastefull as well. Additional empirical data for PEKO benchmarks are given in Table 1 and can be compared to results on benchmarks from the proprietary Cadence-Capo suite [13] shown in Table 5. `Dragon` and `Feng Shui` apparently mishandle multiple sub-rows in a row split by a vertical power stripe. Results for different placers on the IBM-Dragon benchmarks [61, 62] are presented in Tables 7 and 8.

For the IBM-Dragon v1 netlists, the anaytical placer `KraftWerk` does not do very well in terms of HPWL. IBM-Dragon v1 benchmarks are constructed from the partitioning netlists released by IBM [7] and all large cells are removed. Because of

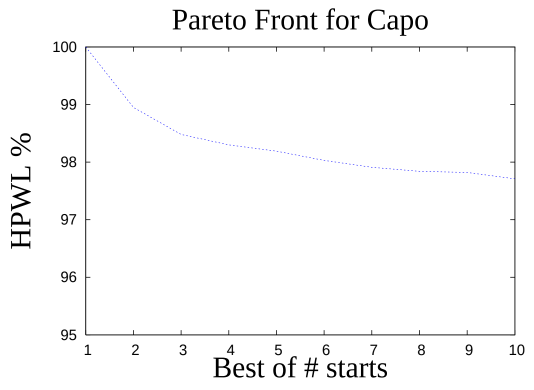


Figure 3: **Pareto front for the randomized min-cut placer `Capo`, based on 100 independent starts on ibm10 benchmark from the `ibm-v1` suite. We plot the expected best-of-N-starts wirelength as the function of the number of starts N. The average-best-of-1-start is calibrated to 100% on the *y*-axis.**

Test	Dragon 2.23				KraftWerk				Capo 8.6			
	Placement				Placement				Placement			
#		+ECO				+ECO				+ECO		
[13]	WL	WL	Time	Route	WL	WL	Time	Route	WL	WL	Time	Route
	*e*8	*e*8	sec	Success	*e*8	*e*8	sec	Success	*e*8	*e*8	sec	Success
1	2.66	3.86	383	No	2.85	2.95	25	No	2.64	2.64	73	**Yes**
2	I	I	I	I	4.01	6.38	185	No	3.08	3.22	268	**Yes**
3	oc	oc	oc	oc	5.9	6.01	149	No	5.55	5.56	158	**Yes**
4	oc	oc	oc	oc	10.5	11.4	95	No	10.5	10.7	163	**Yes**
5	6.02	6.07	2002	**Yes**	t-o	t-o	t-o	t-o	5.68	5.72	326	No
6	a	a	a	a	t-o	t-o	t-o	t-o	155	155	251	**Yes**

Table 5: **HPWL results and runtimes of academic placers on Cadence-Capo benchmarks (2.0GHz Pentium4-Xeon). To legalize placements, we apply Cadence QPlace (SEDSM ver. 5.3) in the -ECO mode and report wirelength before and after. If a placement is legal, no cells are moved. QPlace -ECO moves almost every cell after `KraftWerk`, but the wirelength increase is modest. "I" stands for failure to read the input, "a" denotes failure to produce a placement, "oc" reflects numerous cells outside the core region, "t-o" stands for 24-hour time-outs. `Dragon` aborts in fixed-die mode on one testcases (complaining about the lack of whitespace) and finishes only in variable-die mode on two (test1 and test5). test5 has 33.9K cells and 29.4% whitespace, test6 has 35.5K cells and 0.1% of whitespace.**

Benchmark suites	Dragon [62]	KraftWerk [28]	Capo 8.6 [13]	mPL 2.0 [18]	FengShui 2.0 [65]
Grids	9.19*e*4 (Poor)	3.93*e*4 (Good)	4.68*e*4 (Good)	7.79*e*4 (Med)	7.17*e*4 (Med)
PEKO[18]	4.80*e*6 (Poor)	4.66*e*6 (Poor)	4.09*e*6 (Med)	3.28*e*6 (Good)	3.91*e*6 (Med)
IBM v1[61]	25.50*e*6 (Good)	29.40*e*6 (Med)	27.80*e*6 (Med)	28.90*e*6 (Med)	27.40*e*6 (Med)
IBM v2[62]	40.10*e*7 (Good)	50.8*e*7 (Poor)	40.50*e*7 (Good)	45.40*e*7 (Med)	37.40*e*7 (Good)
IBM v2[62](Route)	Good	Poor	Med	Poor	Poor
Cadence-Capo [13]	Med	Med	Good	N/A	Poor

Table 6: **Average half-perimeter wirelengths achieved by major academic placers on five benchmark suites. We classify performance as Good, Medium or Poor. For IBM-Dragon v2 and Cadence-Capo benchmarks we also compare placers on routability. N/A means that no placements were produced due to input problems, crashes or unsupported features.**

that, the resulting netlists contain disconnected fixed-terminals. We contacted the authors of `KraftWerk` and they speculate that the disconnected fixed-terminals might be the reason for the bad performance. Since the global optimum (in terms of a quadratic cost function) depends on the location of the fixed cells, `Kraftwerk` uses the location of the fixed cells to compute the start solution by default. If there are no fixed cells available (or they have some "pathologic" distribution) one would use a different start solution. `KraftWerk` also does very poorly on the IBM-Dragon v2 netlists and produces placements with large amounts of overlaps. We believe this is also because the IBM-Dragon v2 netlists do not have any fixed terminals. `mPL2.0` also does not do very well on the IBM-Dragon v2 netlists which have cells of non-uniform sizes. The authors of `mPL2.0` admit that `mPL2.0` has problems in processing cells of varying sizes. Grid placements in Figure 2 suggest that (i) it may be difficult for an annealer (in `Dragon`) to place regular structures, (ii) despite good performance, `KraftWerk` seems to ignore connections to fixed terminals. enough, by studying the behavior of `Capo 8.5` grids we were able to improve its performance. A comparison of placer run-times on IBM Dragon v1 benchmarks is presented in Table 4. As seen, `Capo 8.6`, `Feng Shui 1.1` and `mPL 1.2b` are reasonably fast. `Dragon 2.23`, which combines min-cut with annealing algorithms is fairly slow. The latest release of `mPL`, `mPL 2.0` improves wirelength over `mPL 1.2b` but is very slow for the larger benchmarks. A summary comparison of existing placers using data from multiple benchmark suites

Circuit [61]	KraftWerk WL *e*6	+Domino WL *e*6	Capo8.6 WL *e*6	Dragon2.23 WL *e*6	FengShui2.0 WL *e*6	mPL2.0 WL *e*6
ibm01	6.54	5.85	5.03	**4.60**	4.94	5.30
ibm02	20.8	17.8	16.0	**13.7**	14.4	14.9
ibm03	16.2	14.7	13.4	**12.7**	12.9	14.8
ibm04	19.2	18.1	17.9	**16.1**	16.8	18.9
ibm05	40.2	37.8	44.0	**39.0**	37.4	42.7
ibm06	24.7	a	21.9	**19.5**	20.0	23.3
ibm07	38.1	34.4	36.9	**30.9**	30.6	34.5
ibm08	40.4	37.0	38.3	**32.5**	36.7	36.0
ibm09	31.7	29.2	30.1	**28.4**	29.4	31.4
ibm10	66.6	a	60.1	**57.2**	57.6	67.6
Avg	29.4	-	27.8	**25.5**	27.4	28.9

Table 7: **HPWL on 10 out of 18 IBM-Dragon v1 benchmarks. `KraftWerk` placements are legalized by `Domino`. `Domino` aborts on ibm 06 and ibm10. Best results are shown in bold.**

Circuit [62]	KraftWerk WL *e*7	KraftWerk Overlap %	+Domino WL *e*7	Capo8.6 WL *e*7	Dragon2.23 WL *e*7	FengShui2.0 WL *e*7	mPL2.0 WL *e*7
ibm01	7.03	32	6.37	5.49	**5.82**	5.24	6.40
ibm02	21.5	49	t-o	15.9	**15.8**	14.7	16.1
ibm07	51.2	93	t-o	37.0	**35.9**	33.0	40.7
ibm08	46.6	95	t-o	38.4	**38.2**	36.6	42.5
ibm09	42.6	93	t-o	32.2	**32.0**	30.1	38.1
ibm10	76.1	76	a	61.5	**60.2**	56.7	66.1
ibm11	58.0	112	a	48.5	**47.2**	45.9	59.6
ibm12	104.1	27	a	**85.8**	**85.8**	77.5	94.4
Avg	50.8	-	-	40.5	**40.1**	37.4	45.4

Table 8: **HPWL on IBM-Dragon v2 benchmarks (easy suite). The benchmarks have no terminals. `KraftWerk` placements are legalized by `Domino`. `KraftWerk` produces placements with large amounts of overlaps. So, we report the overlaps as a percentage of the layout region area. `Domino` aborts or time-outs(24 hours limit) for most benchmarks. In terms of routability, only `Dragon2.23` produces consistently routable placements on all benchmarks. `Capo8.6` produces unroutable placements for ibm01 and ibm02. `KraftWerk` uniformly distributes whitespace and produces unroutable placements for all benchmarks. `mPL2.0`, `Feng Shui2.0` and `KraftWerk+Domino` pack all cells to the left and produce unroutable placements for all benchmarks.**

is given in Table 6. For all reported results in this paper, the half-perimeter wirelength (HPWL) is measured independently by the wirelength evaluator posted on GSRC bookshelf [16]. These results suggest that `Dragon` is tuned to IBM-Dragon benchmarks, while `Capo` is tuned to Cadence-Capo benchmarks. The original version of `Feng Shui` was tuned to the somewhat outdated `MCNC` benchmarks. Additionally, observe that the `mPL` placer outperforms all other placers only on `PEKO` benchmarks, most likely thanks to its clustering algorithm.

4 Benchmarking for Routability

In the 1980s and early 1990s, works on circuit layout often considered both placement and routing [56, 59]. However, as those tasks became more complex, they were often considered separately in the late 1990s. Such a separation of concerns makes evaluation easier and decreases benchmarking runtime, however, the results may be inconclusive and misleading. Even when placement results are evaluated by running a commercial router [13], this is far from explicit routability improvement during placement [53, 62].

The narrow focus on placement, together with attempts at wirelength prediction, lead to the popularity of wirelength-based metrics that *roughly* model routability, are easy to calculate, and can be integrated into an optimization engine. Errors in such metrics can sometimes be tolerated. Indeed, in variable-die designs using channel-based routing model (common when very few metal layers were available), even a poor placement could be routed, although at potentially high cost. Modern fixed-die designs with high utilization, many metal layers and over-the-cell routing model lead to the new phenomenon of *unroutable placements*. The fact that low-wirelength placements are not necessarily routable motivated recent studies of the routability of different placement methods.

In [13], the `Capo` placement tool was used in a set of experiments on proprietary commercial circuits, for most of which `Capo` placements could be routed using a commercial tool without a great deal of difficulty. While one might conclude that `Capo` placements are *generally routable*, a different conclusion could be drawn from recently published empirical results. In [62], the `Capo` and `Dragon` placement tools were compared using ISPD-98 circuits from IBM [7] that were originally published as hypergraph partitioning benchmarks. The authors of [62] removed large macros, mapped the circuits using an academic 0.18μm cell library from Artisan and added artificial routing grids. The resulting benchmarks are publicly available at [16], and when they are placed with `Capo` the same commercial global router used in [13] frequently fails. We were able to reproduce those experiments with `Capo 8.0` (`Capo 8.6` tends to produce better-routable placements, but not as good as `Dragon`). Additional data are given in Table 5, where six out of seven Cadence-Capo benchmarks are placed by `Capo`, `KraftWerk`, `FengShui` and `Dragon`.

It should not be surprising that optimization of wirelength alone does not guarantee success in routing. Local wiring demand and design characteristics play a significant role. This can have serious repercussions for commercial design teams: the routability of a placement approach may be unknown until actual routing. As for improving placement algorithms, the success or failure gives little insight into what was right or wrong with a placement, or how it may be improved. There is a need metrics that are good at predicting routability, especially at the early stages of placement.

Before suggesting routability metrics for placement, we note that benchmarking of routing itself is problematic. No consensus exists on global routing objectives, and there are no large, widely used public benchmarks. Despite the availability of many well-known placement benchmarks (MCNC, IBM-Dragon and PEKO) in [16] there is nothing comparable for routing. Most papers on global routing use proprietary benchmarks, or test cases that were generated by the authors themselves. Comparisons are frequently made with naïve implementations, or with unnamed commercial tools.

We do not believe that one can easily re-target a layout tool from wirelength-driven metrics to timing and/or power minimization after successful global and detail routing. Therefore we describe an evolutionary transition through a series of simpler metrics that can be incorporated into current work, and provide greater insight into the routability of a placement.[3]

METRIC 1: Simple Congestion Metrics. We explain the reported variations in routability via a detailed examination of results produced by two bisection-based placers, `Capo`[13] and `Feng Shui` [65], and the annealing based placer `Dragon`

[3]More sophisticated models for achievable routing were proposed and validated [35], but the community has yet to produce consistent and independently verifi able results even for simple metrics.

[61]. Table 9 shows half-perimeter wirelengths for the three placers on the IBM-derived placement benchmarks [16]. Besides total wirelength, we decompose the wiring into horizontal and vertical components. *While total wirelengths may only differ by 5% or 10% per benchmark, horizontal and vertical demand may differ by a large margin.*

Bench-	Capo 8.6			Feng Shui 2.0			Dragon 2.23		
mark	WL	% H	% V	WL	% H	% V	WL	% H	% V
	(e6)			(e6)			(e6)		
ibm01	4.97	60	40	4.87	43	57	4.42	53	47
ibm02	15.23	64	36	14.38	49	51	13.57	51	49
ibm03	14.06	59	41	12.84	51	49	12.33	54	46
ibm04	18.13	62	38	16.69	45	55	15.41	51	49
ibm05	44.73	62	38	37.30	53	47	36.38	53	47
ibm06	21.96	59	41	20.27	44	56	20.38	54	46
ibm07	36.06	68	32	31.50	56	44	29.97	55	45
ibm08	37.86	68	32	34.14	59	41	32.20	56	44
ibm09	30.28	62	38	29.86	54	46	28.10	55	45
ibm10	61.25	66	34	57.99	59	41	57.20	62	38
ibm11	46.45	57	43	43.28	46	54	40.77	50	50
ibm12	81.55	62	38	75.91	55	45	71.03	53	47
ibm13	56.47	60	40	54.09	52	48	50.57	53	47
ibm14	128.51	63	37	123.05	55	45	119.80	53	47
ibm15	144.51	65	35	142.90	55	45	134.24	53	47
ibm16	186.04	63	37	185.26	52	48	170.08	54	46
ibm17	278.71	65	35	267.65	54	46	250.56	56	44
ibm18	199.98	65	35	197.85	50	50	188.26	58	42
Avg		63	37		52	48		54	46

Table 9: **Half-perimeter wirelengths for placements by `Feng Shui, Capo`, and `Dragon` on IBM-Dragon v1 benchmarks. The distribution into *horizontal* and *vertical* components reveals large differences – `Capo` produces placements that use significantly more horizontal wiring than vertical; earlier versions of `Feng Shui` also exhibited this behavior. The use of routing resources on metal layers is a key routing-related concern.**

ASIC routing is normally performed using "preferred direction" wiring; clearly, `Capo` targets significantly higher horizontal demand and significantly lower vertical demand. Early versions of `Feng Shui` also had heavily biased routing demands. A bias of this type may sometimes be beneficial: industrial benchmarks often have more horizontal routing resources than vertical. If routing fails for `Capo`, but not for `Dragon` or `Feng Shui`, the router likely cannot find a location where a wire can travel horizontally. If routing fails for `Dragon` or `Feng Shui`, but not for `Capo`, it is likely that the router cannot find a location where a wire can travel vertically.[4] The difference in routing demand also suggests how to deal with interconnect layers. If the number of metal layers is odd or differences in routing pitches bias routing supply in one direction, the placer should bias the routing demand accordingly.

Evaluating vertical and horizontal wirelength is easy and helps explain apparently contradictory results. Commercial tools report the two numbers and their sum. Academic tools should do the same.

Extending the simple separation between horizontal and vertical components is a metric similar to channel density. Sweeping through the layout either vertically or horizontally, one can track horizontal and vertical routing demand. "Best-case" and "worst-case" congestion levels for the H and V routing layers can be found. When horizontal and vertical placement densities are compared for `Feng Shui`, `Capo`, and `Dragon`, the overall results are similar to those in Table 9.

Our final suggestion for "simple" congestion metrics are those based on "probabilistic" routing models [53, 44, 38] as follows.

[4]If routing succeeds, it may take an unusually long time, indicating layout problems. To avoid congestion, a route may need to detour. Such detours increase path lengths, substantially slowing down maze search in modern routers.

- The core area is decomposed into a regular grid of routing tiles.
- Each signal net is decomposed into steiner or spanning trees.
- The "probability" that a given tree edge uses a given tile is computed based on fast combinatorial enumeration of shortest paths.

An open-source implementation of probabilistic congestion maps from [44] is distributed with the `Capo` placer in [16] and can produce picture files as well as scripts for Matlab and gnuplot — see Figure 4. This estimation method is reasonably fast, can obtain results that are close to those of global routing tools [53, 38]. These estimates should be used with caution, however: good global routing tools may introduce slight routing detours to eliminate congestion problems. Probabilistic models might be considered pessimistic; if the estimates are used to influence the placement process, we may be addressing problems which do not actually exist, and suffer unnecessary wirelength increases.

METRIC 2: Global (and Detail) Routing. Presently, relatively few routers are available publicly. Global routers `Labyrinth` [39] and the `Force-Directed Router` [47] are both downloadable from [16] in source code (in C++ and Java respectively), but their behavior on large circuits may not be representative of commercial routers. Some research groups use commercial tools [13, 62], most frequently Cadence `WarpRoute`. However, commercial tools are impossible to tweak and difficult to integrate with. In particular, commercial tools typically do not save global routing results (which would be convenient for evaluating global placement) but rather offer a monolithic global+detail routing optimization. Furthermore, commercial routers may obscure results by performing sophisticated optimizations. To summarize, we believe that an open infrastructure for global routing should be developed by academic researchers and populated with open-source routers of reasonable quality, tested against commercial tools (similarly to how major academic global placers have been tested). A fast global router can be then embedded into a placer [52] as an estimator.

5 Toward Open Benchmarking for Timing-driven Placement

The development of scalable, powerful and robust algorithms for circuit delay minimization during placement is a key challenge in Circuit Layout. It is mentioned regularly in the requests from industry and government funding agencies, but few replicable results have been reported in the literature. While we discuss timing, parallels can be made with power minimization. Barriers to research in timing-driven placement can be summarized as follows.

- **Lack of non-trivial placement benchmarks** with enough information to perform accurate timing analysis. The MCNC benchmarks which have signal direction information use an extremely simple and outdated timing model. Meanwhile, benchmarks derived from academic work are viewed by industrial groups as small and meaningless. "Synthetic" benchmarks are criticized for not accurately modeling "real" circuits.
- **Accurate circuit-level timing analysis is non-trivial**, and accurate device-level timing analysis is computationally expensive.
- **Actual design parameters are closely guarded industrial secrets**, and profoundly influence interconnect delay.

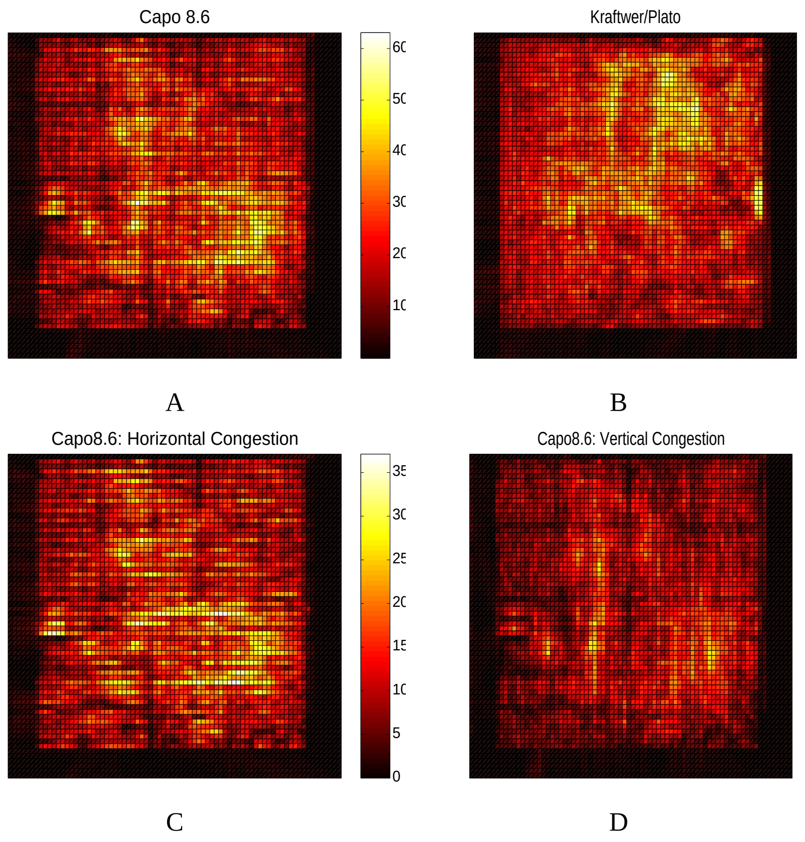


Figure 4: **Probabilistic congestion maps for Cadence-Capo test3 (20.5K cells). The `Capo` placement (A) has higher peak congestion, but lower average congestion than `KraftWork` (B). Cadence WarpRoute succeeded in both cases: 6 violations for `Capo` and 123 for `KraftWerk`. For Capo placements the horizontal congestion (C) is higher than the vertical congestion (D) The plotter program is available in the GSRC Bookshelf under Placement Utilities.**

For the available benchmarks, differences in interpretation that have plagued wirelength-based placers [46] are more problematic in the context of timing optimization. The timing-driven annealing-based placer from [59] reports the longest path delay 798*ns* for the MCNC benchmark `avqsmall`. In [28], the longest path was improved to only 80*ns*. A quadrisection driven placer [33] reported a result of 71*ns*, and most recently, a result of 59.6*ns* was reported [51]. The improvement in delay for the same circuit *by more than a factor of 10* seems beyond belief, especially considering that the approach of [59] was by no means naïve and their placer implementation has been validated independently. Also note that the `avqsmall` circuit was released in 1989, and clock frequencies of 16.6MHz were not realistic for standard-cell ASICs at that time. At this time, while interconnect delay was important, it by no means dominated system delay. Even if all interconnect delay was eliminated, it is unlikely that the delay of the longest path could be affected to this extent. Further investigation revealed that some path delays reported in [51] are smaller than the sum-of-gate-delays reported in [32] — for the testcase `fract` [51] computes a path delay 11.91*ns*, while [32] produces a lower bound of 18.5*ns* by entirely ignoring interconnect delays.

Compounding the different interpretations of the benchmarks is the issue of what would be an appropriate timing-driven objective. One could suggest that the setup-slack (the difference between path arrival time and path required time) reported by a static-timing-analysis (STA) engine should be the final arbiter of the “goodness” of a TDP. Indeed, among recent TDP papers [51, 32, 22, 36, 63, 41] one half [22, 36, 63] do just that. However, some groups may find it difficult to obtaining valid timing constraints, gate models (delay library) and an appropriate technology-file to correctly compute the setup-slack. To overcome the obstacles in using an industrial STA engine, authors frequently report “path-delays” through some gate-delay computation coupled with internally developed STA engines. Such timing analyzers can be simplified by ignoring path-exceptions, multiple-clock domains, delays on primary I/O, and gate-delay modeling and net modeling details. The impact of slope (signal-transition time) on gate-delay is typically ignored, likely making path-delay results erroneous as shorter paths with long nets appear more critical than paths with more stages of logic.

At the heart of TDP lies an inherent compromise between optimization and simulation. Ideally each decision made by the placement engine must be guided by exact setup-slack. However, even one pass of an accurate STA may be prohibitively slow in some cases.

While various design considerations make it extremely difficult to evaluate timing accurately, academic works typically address geometric and graph-theoretic aspects that are also challenging for commercial tools. Indeed, signal paths that detour a lot typically have greater delay than “straight” paths. A simple but non-trivial objective function is given by the total geometric path length (gate delays can be added easily to such formulations). To this end, algorithms that directly attempt to “straighten” critical paths by optimizing geometric path lengths have been proposed [36] and extended to more realistic delay objectives. During such optimization they must ensure that sub-critical paths do not overtake currently-critical paths. These algorithms need only the infrastructure to evaluate the criticality of paths and are accessible to academic groups.

Physical synthesis is a synergistic attempt at design closure via simultaneous placement and logical transforms [27, 48, 45]. While interesting work on Physical Synthesis, with empirical results, already appeared at conferences [42], no replicable timing results are given. In Physical Synthesis, concerns about ignoring transition time are alleviated by interleaving placement transforms with calls to netlist buffering [27]. However, this raises two additional concerns: the netlist or gate sizes may change from one iteration to the next, and regions of the chip may become over utilized, thus requiring powerful legalization methods. An alternative method is to perform the placement optimization within a ”virtual buffering” mode [48]. This allows the placement engine to operate on a constant netlist (buffers are not inserted) with a timing analysis mode that minimizes excessive slope effects and correctly accounts for buffer delays. In a gain-based synthesis environment [45] this problem is converted into the task of maintaining the gain on each cell. While the netlist does not change, the sizes of the cells may change (to maintain gain), leading to the need for strong legalization techniques. In either case, transition-time effects may lead to 5-10% larger gate area and further challenges for the TDP engine. Timing optimizations during placement must also look at the effects of physical synthesis on the design. Results in Table 10 show that physical synthesis transforms after placement significantly affects the timing of a design.

It may be unrealistic to develop a Physical Synthesis environment in academia in the near future. Moving beyond the simple task of timing-driven placement is challenging, much of the basic infrastructure needed to support academic research (i.e., static timing analysis and high quality benchmarks) is not currently available. However, we do envision a set of benchmarks with valid timing constraints, multiple clock-domains and of representative size. This requires access to gate-delay libraries (`.lib`) and technology files (LEF). Finally, there needs to be a way to independently verify the timing results of the placements. Some necessary infrastructure may be provided by recent efforts at Si^2 that resulted in downloadable software such as OLA [50] and OpenAccess [49], but path-based STA is still missing. The descriptions below are adapted

Design	# Nodes	Slack (ns)		
		Initial(Total -ve)	Sized(Total -ve)	Buffered(Total -ve)
D1	22253	-2.75(-508)	-2.17(-512)	-0.72(-21)
D2	89689	-5.87(-10223)	-5.08(-9955)	-3.14(-5497)
D3	99652	-6.35(-8086)	-5.26(-5287)	-4.68(-2370)
D4	147955	-7.06(-7126)	-5.16(-1568)	-4.14(-1266)
D5	687946	-8.95(-4049)	-8.80(-3930)	-6.40(-3684)

Table 10: **Effect of physical synthesis on timing. Designs are proprietary benchmarks as no public infrastructure currently exists for physical synthesis. In the context of sizing and buffering optimizations after placement, we report the worst slack and the sum of negative slack along all critical paths, before and after sizing and buffering. We do not intend to show competitive results for these benchmarks, but rather emphasize that physical synthesis significantly affects timing evaluation.**

from [50, 49].

OLA is an Application Procedural Interface (API) that can be used by EDA tools for the determination of cell and interconnect characteristics of very deep submicron ICs. OLA is an extension to the Standard for Delay and Power Calculation System, the IEEE 1481-1999 standard. Target applications include timing-driven placement and routing, and OLA attempts to eliminate inconsistent timing data between different EDA tools by using the library vendor's ”golden” delay calculator in all OLA compliant tools.

The OpenAccess API is a C++ program interface to IC design data. The associated reference database is a technology donation from Cadence Design Systems, who is also a member of the OpenAccess Coalition. The API and the reference implementation provide a high performance, high capacity electronic design database with architecture designed for integration and fast application development. Access to the reference database source code is provided to allow companies and academic institutions to contribute to future database enhancements and add proprietary extensions. Cadence Design Systems is planning to migrate all of its tools to the OpenAccess database by the end of 2003. Bridges to proprietary design databases used at Synopsys and IBM are currently under development by those companies and third-party vendors.

6 Conclusions

To seriously address the huge sub-optimality of existing placement tools [31, 25, 18], one needs to ascertain improvements on industrial circuits. However, published empirical data show that even when two research groups use the same source data, *there are often differences of interpretation, resulting in incompatible numbers and no useful conclusions made from the data.* For example, timing-driven placement benchmarks posted in Verilog [22] prevent reliable comparisons to published numbers, e.g., in [63]. To remedy such incomplete benchmarks, the Vertical Benchmarking project at CMU [55] offers multiple representations of the same design. However their benchmarks still do not have sufficient timing data. On the positive side, recent placement benchmarks better agree in terms of row spacing, pin positions, etc and researchers are more conscientious about such design aspects [46].

New technology generations and prevailing design methodologies may influence various layout features, potentially affecting comparisons between optimization algorithms. Thus, experimental protocols should reflect the nature of layouts (e.g., routability is more important in fixed-die layout) and algorithms (e.g., the results of a randomized algorithm can be averaged). To this end, a number of lessons from placement benchmarking are summarized below:

1. Evaluation methods must be explicit to leave minimum room for misinterpretation. Simple open-source evaluation tools should be used to verify the accuracy and correctness of any published result. For example, open-source plotters of placement and congestion, as well as evaluators of wirelength and congestion are distributed with the `Capo` placer in the GSRC Bookshelf [16]. Linux and Solaris binaries are posted in the *Placement Utilities* slot. Benchmarks should be explicit too, and no preprocessing by user should be assumed. The same input files should be used for all tools compared. When conversion cannot be avoided, standard publicly available converters should be used — we posted such converters in the *Placement Utilities* slot of the GSRC Bookshelf.

2. Raw experimental results are very useful and should be posted on-line. This simplifies the verification of results, and may lead to insights into what a tool did "right" or "wrong" on various problems. In the same vein, the version of each tool should be reported (it's easy!) or at least the time when each tool was downloaded and the source. This can resolve potential confusion about outdated versions of public EDA tools.

3. Visualizations, especially on small benchmarks, help identifying and diagnosing problems. In the course of our work, the performance of `Capo`, `Feng Shui` and `mPL` was improved through step-by-step analysis of placement process on grid benchmarks. A bug in `Dragon 2.20` fixed in `Dragon 2.23` is illustrated in Figure 1. We recommend placement results be sanity-checked by plotting (are all cells in the core area, do macros overlap?).

4. Regressions are common when bugs are fixed. Last-minute placer bugfixes sent to us by developers occasionally produced worse results than prior versions. For example, `mPL 1.2b` placed the PEKO01 benchmark with wirelength 1.17e6 versus 1.09e6 achieved by `mPL 1.2`. We suspect that this deterministic implementation uses a randomized algorithm with a fixed seed, making the results somewhat chaotic. One could expose randomization, as in `Capo` and `Dragon`, to stabilize evaluation via averaging [10].

5. Open-source tools are very valuable as they enable interesting experiments via slight modifications. For example, terminal propagation is not described adequately in placement literature, and the best way to learn successful approaches to it is to look at open-source codes [16]. The same applies to many implementation details of high-performance min-cut partitioning algorithms [12]. Open source also lowers barriers to entry and leads to more meaningful research work. Instead of implementing new parsers and basic algorithms, researchers should focus on key aspects of EDA tools. To accelerate this culture change, we suggest that conference program committees and reviewers favor submissions accompanied by open-source implementations, or at least publicly available binaries. However, making this a requirement seems too harsh.

6. Researchers should use (and be encouraged to use) open benchmarks whenever possible. This allows others to replicate, confirm and study published results in detail, and may also facilitate the publication of negative results, which still remains a challenge. Independent evaluation may also point out biases in experimental protocols, and can improve the overall quality of research output in our field. However, it is likely that no open benchmarks are available for experiments with emergent design aspects. To this end, results for proprietary netlists may still be useful, as their publication suggests an opportunity for new public benchmarks.

7. Despite the overall preference for real design benchmarks, artificial testcases with known optimal solutions [31, 18] are becoming popular. Instead of known optimal solutions, bounds on optimal costs will do. Such benchmarks (BEKU) are proposed in [23] for min-cut hypergraph partitioning.

As we focus on more difficult problems, the community must support open benchmarking and tool availability, otherwise we cannot expect much progress.

Benchmarking For Routing Tools.

With variable-die channel-based standard-cell designs, comparing global routing tools was relatively easy. Channel density can be computed directly, and channel routing tools can often achieve the lower-bound target. Feed-throughs are inserted in cell rows; given the length of the longest row and the total channel density, we can obtain a very accurate estimate of chip area after detail routing.

Fixed-die, multilayer over-the-cell global routing is more difficult to evaluate because detail routing is non-trivial and must be de-coupled. Technology-specific constraints, e.g., antenna rules, make it impossible to predict successful routing for dense designs [54].

Reasonable metrics for global routers were proposed in [53, 39]:

- Each edge of the global routing graph has a fixed maximum capacity; this is a hard physical constraint, and any routing which exceeds this is infeasible.
- When routing demand is below capacity, successful detail routing is more likely. In [53], 70-80% was proposed as a good objective. If a routing solution exceeds this level for a given edge, the edge is “over capacity”. Reducing the total amount by which all edges exceed the target capacity is a reasonable goal.
- If capacity constraints are met, reduce the total wirelength.

A number of global routing benchmarks were made available in [16] by the authors of [39]. As the community moves toward wider usage of benchmarks, these can be suggested as a reasonable next step. For detail routing, very little is available for benchmarking. Only a few research groups are actively working on detail routing tools, and the problem is made extremely complex due to differing design rules, numbers of routing layers, and performance objectives such as crosstalk, delay, and even lithography related issues.

Delay, Power and Temperature.

Incompatible data published for the MCNC benchmarks `fract` and `avqsmall` suggest wide-ranging interpretations and modeling of signal delay, rise and fall times, etc. Given a placement and routing solution, two researchers may come up with “delay” or “power” numbers that are off by an order of magnitude. If the community is to actively pursue timing-, power- and temperature-driven layout, common frameworks are required to evaluate these objectives. We hope that [49, 50] may provide such frameworks. As for public benchmarks with enough information to evaluate signal delay, we are currently negotiating with our colleagues in the industry and hope to post new benchmarks in the GSRC Bookshelf [16]. However, detailed comparisons including delay will require much more effort and finesse than the comparisons presented in this paper.

Wider Benchmarking Context.

When we consider layout problems identified in "research needs" documents from funding agencies, many areas appear in need of benchmarks, even to reliably verify results of one's research by experiment. We feel that aside from identifying important problems the community must developed evaluation methods and agree upon them. To be specific, we mention several sample areas where benchmarking could help. *Mixed digital-analog design for SOC* and *3-dim integration* raise new layout issues. *The X-routing architecture* with 45-degree wiring may affect basic placement and routing algorithms. *Multiple-voltage* systems are now being developed to reduce power consumption without sacrificing performance. Public benchmarks are lacking for such non-traditional designs despite their relevance to next-generation circuitry. Also *physical verification, reliability and yield issues* are becoming more important every year.

In summary, we propose that the physical design community adopt standards for empirical evaluation and best practices similar to those in the placement community. This could improve the quality of on-going work on circuit layout as well as the interaction among researchers, practitioners and funding agencies.

Acknowledgments

This work was supported by the Semiconductor Research Corporation, the Gigascale Silicon Research Center, an IBM University Partnership award, an IBM Faculty Partnership award, and equipment grants from Intel. We would also like to thank to Prof. Frank Johannes (TU Munich), Prof. Jason Cong, Prof. Joseph Shinnerl, Min Xie (UCLA), Prof. Andrew Kahng (UCSD) and Xiaojian Yang (Synplicity) for technical discussions and help with placement tools. Sherief Reda (UCSD) pointed out mistakes in Table 5 of the conference (ISPD 2003) version of this paper. We report corrected data in Table 9.

References

[1] S. N. Adya and I. L. Markov, "Consistent Placement of Macro-Blocks using Floorplanning and Standard-Cell Placement", *ISPD* 2002, pp. 12-17.

[2] S. N. Adya, I. L. Markov and P. G. Villarrubia, "On Whitespace and Stability in Mixed-Size Placement and Physical Synthesis", *ICCAD* 2003.

[3] A. Agnihotri, M. C. Yildiz, A. Khatkhate, A. Mathur, S. Ono and P. H. Madden, "Fractional Cut: Improved Recursive Bisection Placement", *ICCAD* 2003.

[4] C. J. Alpert, G.-J. Nam and P. G. Villarrubia, "Free Space Management for Cut-Based Placement", *ICCAD* 2002, pp. 746-751.

[5] C. J. Alpert and A. B. Kahng, "Recent direction in netlist partitioning: A survey", *INTEGRATION: The VLSI J. vol. N 19*, 1995, pp. 1-81.

[6] C. J. Alpert, J.-H. Huang and A. B. Kahng,"Multilevel Circuit Partitioning", *DAC* 1997, pp. 530-533.

[7] C. J. Alpert,"The ISPD98 Circuit Benchmark Suite," *ISPD* 1998, pp. 80-85. `http://vlsicad.cs.ucla.edu/~cheese/ispd98.html`

[8] C. J. Alpert, A. E. Caldwell, A. B. Kahng, I. L. Markov, ”Hypergraph Partitioning With Fixed Vertices”, IEEE Trans. on CAD, vol. 19, no. 2, 2000, pp. 267-272.

[9] M. R. Beasley, S. Datta, H. Kogelnik, H. Kroemer, and D. Monroe. “Report of the Investigation Committee on the Possibility of Scientific Misconduct in the Work of Hendrik Schon and Coauthors,” 2000. `http://www.lucent.com/news_events/researchreview.html`

[10] A. E. Caldwell, A. B. Kahng, A. A. Kennings, and I. L. Markov, “Hypergraph Partitioning for VLSI CAD: Methodology for Reporting, and New Results,” *DAC* 1999, pp. 349-354.

[11] A. E. Caldwell, A. B. Kahng and I. L. Markov, “Improved Algorithms for Hypergraph Bi-partitioning,” *ASPDAC* 2000, pp. 661-666.

[12] A. E. Caldwell, A. B. Kahng and I. L. Markov, “Design and Implementation of Move-based Heuristics for VLSI Hypergraph Partitioning”, *ACM Journal of Experimental Algorithms, vol. 5*, 2000
`http://www.jea.acm.org/volume5.html`

[13] A. E. Caldwell, A. B. Kahng and I. L. Markov, “Can Recursive Bisection Alone Produce Routable Placements?”DAC 2000, pp.477-82.

[14] A. E. Caldwell, A. B. Kahng, I. L. Markov, “Optimal Partitioners and End-case Placers for Standard-cell Layout”, *IEEE Trans. on CAD, vol. 19(11)* 2000, pp. 1304-1314

[15] A. E. Caldwell, A. B. Kahng, I. L. Markov, “Hierarchical Whitespace Allocation”, *IEEE Trans. on CAD, vol. 22(11)* 2003.

[16] A. E. Caldwell, A. B. Kahng, I. L. Markov, “VLSI CAD Bookshelf” `http://vlsicad.eecs.umich.edu/BK`

[17] T. Chan, J. Cong, T. Kong, and J. Shinnerl, “Multilevel Optimization for Large-scale Circuit Placement,” *ICCAD* 2000, pp. 171-176.

[18] C. C. Chang, J. Cong and M. Xie, “Optimality and Scalability Study of Existing Placement Algorithms,” *ASP DAC* 2003, pp. 621-627.

[19] C.-C. Chang, J. Cong, and X. Yuan, “Multi-level Placement for Large-Scale Mixed-Size IC Designs,” *ASPDAC* 2003, pp. 325-330.

[20] T. Kong, J. R. Shinnerl and K. Sze, “An Enhanced Multilevel Algorithm for Circuit Placement,” *ICCAD* 2003.

[21] H. Chen, C. Qiao, F. Zhou and C.K. Cheng, “Refined Single Trunk Tree: A Rectilinear Steiner Tree Generator For Interconnect Prediction,” *Intl. Workshop on System-Level Interconnect Prediction (SLIP)* 2002, pp. 85-89.

[22] Y.-C. Chou and Y.-L. Lin, “A Performance-driven Standard Cell Placer based on a Modified Force-directed Algorithm,” *ISPD* 2001, pp. 24-29.

[23] J. Cong, M. Romesis, and M. Xie, “Optimality, Scalability and Stability Study of Partitioning and Placement Algorithms”,*ISPD* 2003, pp. 88-94.

[24] J. Cong and J. R. Shinnerl, “Multi-level Optimization in VLSI CAD,” *Kluwer*, Boston, 2002.

[25] W. J. Dally and A. Chang, “The Role of Custom Design in ASIC Chips”, *DAC* 2000, pp. 643-647.

[26] K. Doll, F. M. Johannes and K. J. Antreich, “Iterative Placement Improvement By Network Flow Methods”. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol.13, (no.10)*, Oct. 1994. pp. 1189-1200.

[27] W. Donath et al., “Transformational Placement and Synthesis”, *DATE* 2000, pp. 194-201.

[28] H. Eisenmann and F. M. Johannes, “Generic Global Placement and Floorplanning”, *DAC* 1988, pp. 269-274.

[29] C. M. Fiduccia, R. M. Mattheyses, “A Linear-Time Heuristic For Improving Network Partitions”, *DAC* 1982, pp. 171-181.

[30] S. Goto, “An Efficient algorithm for the Two-Dimensional Placement Problem in Electrical Circuit Layout,” *IEEE Trans. on Circuits and Systems, vol. 28 no. 1*, 1981, pp. 12-18.

[31] L. Hagen, J. H. Huang, and A. B. Kahng, “Quantified Suboptimality of VLSI Layout Heuristics”, *DAC* 1995, pp. 216-221.

[32] W. Halpin, C. Y. Roger Chen, and N. Sehgal, “Timing driven placement using physical net constraints,” *DAC* 2001, pp. 780-783.

[33] D. J.-H. Huang and A. B. Kahng. “Partitioning based standard cell global placement with an exact objective,” *ISPD* 1997, pp. 18-25.

[34] S.-W. Hur and J. Lillis, “Mongrel: Hybrid Techniques For Standard Cell Placement,” *ICCAD* 2000, pp. 165-170.

[35] A. B. Kahng, S. Mantik and I. L. Stroobandt, “Requirements for Models of Achievable Routing,” *ISPD* 2000, pp. 4-11.

[36] A. B. Kahng, S. Mantik and I. L. Markov, “Min-max Placement For Large-scale Timing Optimization” *ISPD* 2002, pp. 143-148.

[37] G. Karypis, R. Agarwal, V. Kumar, and S. Shekhar, “Multilevel Hypergraph Partitioning: Applications in VLSI Design”, *DAC* 1997, pp. 526-529.

[38] P. Kannan, S. Balachandran, and D. Bhatia, “On Metrics for Comparing Routability Estimation Methods for FPGAs,” *DAC* 2002, pp. 70-75.

[39] R. Kastner, E. Bozogzadeh, and M. Sarrafzadeh, “Predictable Routing,” *ICCAD* 2000, pp. 110-113.

[40] J. Kleinhans, G. Sigl, F. Johannes and K. Antreich, “GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization”, *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 10 (3), March 1991. pp. 356-365.

[41] T. Kong, “A Novel Net Weighting Algorithm for Timing-Driven Placement”, *ICCAD* 2002, pp. 172-176.

[42] P. Kudva, A. Sullivan and W. E. Dougherty, ‘ ‘Metrics for Structural Logic Synthesis”, *ICCAD* 2002, pp. 551-556.

[43] E. Lerner, “Fraud Shows Peer-review Flaws,” *The Industrial Physicist, 8(2)* 2002.

[44] J. Lou, S. Krishnamoorthy, H. S. Sheng, “Estimating Routing Congestion using Probabilistic Analysis,” *ISPD* 2001, pp 112-117.

[45] Magma Design Automation Inc., "White Papers," `http://www.magma-da.com/whitepapers.html`

[46] P. H. Madden, "Reporting of Standard Cell Placement Results," *IEEE Trans. on CAD, 21(2)* Feb. 2002, pp. 240-247.

[47] F. Mo, A. Tabbara, and R. K. Brayton, "A Force-directed Maze Router," *ICCAD* 2001, pp. 404-408.

[48] Monterey Design Systems, "Dolphin," `http:// www.mondes.com/products/dolphin.htm`

[49] Open Access, `http://www.si2.org/openaccess/`

[50] Open Library Architecture (OLA), `http://www.si2.org/OLA/`

[51] S.-L. Ou and M. Pedram, "Timing-driven Placement based on Partitioning with Dynamic Cut-net Control", *DAC* 2000, pp. 472-476.

[52] P. N. Parakh, R. B. Brown, K. A. Sakallah, "Congestion Driven Quadratic Placement", *DAC* 1998, pp. 275-278.

[53] A. Rohe and U. Brenner, "An Effective Congestion Driven Placement Framework," *ISPD* 2002, pp. 6-11.

[54] L. Scheffer and E. Nequist, "Why interconnect prediction doesn't work," *Intl. Workshop on System-Level Interconnect Prediction (SLIP)* 2000, pp. 139-144.

[55] H. Schmit, "Vertical Benchmarks," `http://www.ece.cmu .edu/˜herman/html/benchmark_slot.html`

[56] G. Sigl, K. Doll and F. M. Johannes, "Analytical Placement: A Linear or Quadratic Objective Function?" *DAC* 1991, pp. 57-62.

[57] The Standard Performance Evaluation Corporation (SPEC), "SPECmark benchmarks," `http://www.specbench.org/`

[58] D. Stroobandt, "A Priori Wire Length Estimates for Digital Design," 324 pages, *Kluwer*, ISBN 0-7923-7360-X, 2001.

[59] W. Swartz and C. Sechen, "Timing-Driven Placement For Large Standard-Cell Circuits," *DAC* 1995, pp. 211-215.

[60] J. Vygen, "Algorithms for Large-Scale Flat Placement", *DAC* 1997, pp. 746-751.

[61] M. Wang, X. Yang and M. Sarrafzadeh, "Dragon2000: Standard-cell Placement Tool for Large Industry Circuits," *ICCAD* 2000, pp. 260-263.

[62] X. Yang, B.-K. Choi and M. Sarrafzadeh, "Routability Driven White Space Allocation for Fixed-Die Standard-Cell Placement," *ISPD* 2002, pp. 42-50.

[63] X. Yang, B.-K. Choi and M. Sarrafzadeh, "Timing-Driven Placement using Design Hierarchy Guided Constraint Generation," *ICCAD* 2002, pp. 177-184.

[64] X. Yang, B-K. Choi, and M. Sarrafzadeh, "A Standard-Cell Placement Tool for Designs with High Row Utilization," *ICCD* 2002, pp. 45-47.

[65] M. C. Yildiz and P. H. Madden, "Improved Cut Sequences for Partitioning Based Placement," *DAC* 2001, pp. 776-779.

Appendix A: Available Benchmarks

Artificially-generated netlists may be useful for regression-testing and sanity-checking, but placement algorithms are typically validated and compared using netlists derived from real designs. Material in Section 2 explains why.

MCNC benchmarks date back to late 1980s; they are small and outdated [7]. The multitude of their interpretations makes any reported numbers meaningless. Placement variants of MCNC benchmarks converted to the GSRC Bookshelf format are still available in [16] and are sometimes referred to as GSRC-MCNC benchmarks. The GSRC-MCNC benchmarks have fixed terminals.

In 1998 IBM released 18 netlists with 10K-220K modules as benchmarks for hypergraph partitioning [7]. Despite all design information being sanitized, two out of twenty original netlists were not cleared for public release. However, the remaining 18 benchmarks were soon adapted to placement.

IBM-Dragon [61, 62] benchmarks come with the `Dragon` placer discussed below and are referred to as IBM-Place by their authors. Several suites of these benchmarks are available on-line. The first suite (v1) released in 2000 sizes cells based on the node area in the IBM circuits, and removes all large cells. Because of that, the resulting netlists contain disconnected fixed-terminals. The second suite (v2) maps cells to an Artisan library and adds enough routing information to run Cadence WarpRoute on a given placement. IBM-Dragon v2 benchmarks have fixed terminals, but they are all placed at the location (0,0) and are not connected to anything. These benchmarks come in separate versions (i) the bookshelf format, (ii) LEFDEF for Dragon and (iii) LEFDEF for `Capo`. While the authors explain the difference between (ii) and (iii) by problems with LEFDEF parsers, we suggest that it is better to run multiple placers on the same input files to avoid introducing hidden discrepancies in results. A suite of timing-driven placement benchmarks recently posted by the authors of `Dragon` is not related to their previous benchmarks, but rather derived from the ISPD 01 suite [22] discussed below. No timing constraints are given (February 03), and other important information is missing.

IBM-Floorplacement [1] benchmarks are also derived from the IBM netlists, but include all original macros. They are available in LEF/DEF and in the GSRC format. These benchmarks have connected fixed terminals corresponding to the original partitioning netlists released by IBM. The fixed terminals are placed randomly on the edges of the core region. Improved wirelengths have been reported recently [19]. Several known placers produce many cell overlaps on these benchmarks and cannot fit all macros inside the layout areas. Therefore we recommend visualizing placements and checking for overlaps before reporting wirelength.

PEKO [18] benchmarks reflect the net-degree distribution of the 18 IBM netlists, but are otherwise generated artificially, with randomization, offering astronomically large irregular netlists to test the scalability of placers [31]. PEKO stands for Placement Examples with Known Optimal wirelength. In particular, in optimal placements each net independently achieves the smallest possible wirelength, making all wires local. The authors conclude that existing placers are 40-100% away from optimal solutions. However, there is a lingering concern that PEKO benchmarks are not representative of industry circuits. PEKO benchmarks have 15% whitespace and come in two suites. The PEKO benchmarks have no fixed terminals and all the standard-cells are of the same size.

PEKU [23] benchmarks are Placement Examples with Known Upper bounds for optimal solutions. Unlike in PEKO benchmarks, not all nets are short in known good placements. In fact, *all* nets are long, which makes these netlists unrealtistc. However, a hybrid of PEKO and PEKU constructions allows one to achieve a prescribed percent of short nets. The experiments in [23] suggest that the relative performance of known placers depends on that paramater.

Grids with four fixed vertices and n^2 1x1 movables, such as the one in Figure 2, are used in our work to test the behavior of placers on datapath-like circuits, on which many commercial layout tools perform poorly [25]. A simple induction argument shows that there is only one optimal placement for each such "netlist," where each of $2n^2 - 2n + 4$ nets has length 1 (cf. PEKO benchmarks). More importantly, sub-optimalities can be visualized to drive debugging efforts. The benchmarks are on-line at [16].

Vertical benchmarks [55] created at CMU attempt to remedy the lack of design information in public circuit benchmarks. They provide multiple representations of real circuits at different stages of design process, including have non-trivial layout features, such as fixed macros. However, most of those circuits have under 50K cells. As of February 2003, details are missing to evaluate signal delay and there are no definitions of timing constraints, clocks or acceptable transition times. The netlists are mapped to a $0.35\mu m$-library, thereby making interconnect effect negligible.

Non-benchmarks. The ISPD 01 suite from [22] is available in a hierarchical (not flattened) gate-level Verilog format. There are no timing constraints or gate libraries available. There are multiple top-level signals with prefix "CLK", and it is not clear how clock nets are represented. The authors suggest that a series of proprietary tools be applied to their benchmarks before and after timing-driven placement. However, because of differences in tools (versions, options, cell libraries etc) such pre-processing may lead to different numbers even if the same timing-driven placer is used.

Appendix B: Placers

We describe large-scale standard-cell placers available to multiple research groups; the order reflects when the tools were developed and reported in publications. All placers we use except for `KraftWerk` directly support the GSRC Bookshelf [16] placement format that is considerably simpler than LEF/DEF.[5] Some of them also support subsets of Cadence LEF/DEF, but input problems are common with industrial circuits.[6]

KraftWerk [28] is a force-directed placer. Yet, rather than moving one cell a time, it solves the Poisson equation (a PDE from mathematical physics). This analytical algorithm often leaves cell overlaps and requires a separate follow-up legalization step. However, overlaps are typically well distributed over the core area and can be removed by a simple built-in legalization algorithm, perhaps at a cost of increased wirelength. This simple legalization may fail and the user is advised to call the variable-die `Domino` [26] detail placer shipped with `KraftWerk`. `Domino` is based on network-flow algorithms. Both tools target fixed-die layout and tend to distribute whitespace fairly uniformly. `KraftWerk` is deterministic.

Not having access to source codes of `KraftWerk` or `Domino`, we obtained Linux executables in November 20002 directly from the authors, who mentioned that so far `KraftWerk` has not improved wirelength minimization beyond the original 1998 version. The binary for `KraftWerk` is called `Plato`.

Capo [13, 14] is a global fixed-die placer based on recursive min-cut bisection. It uses a built-in multi-level Fiduccia-Mattheyses partitioner [11] written from scratch for this application. All source codes are available in [16]. `Capo` has a

[5]In our experiments, `Feng Shui 1.2` misinterpreted row information in PEKO01 benchmarks. Newer versions fi x that problem.
[6]`Capo` is now shipped with Cadence's offi cial LEF/DEF parser, and the users can switch to it from the default native parser.

built-in LEF/DEF interface and has been tuned on proprietary benchmarks from Cadence, with successful routing in mind (using WarpRoute or any other tool). `Capo` uniformizes whitespace to generically improve routability, but may produce unroutable placements for challenging circuits.

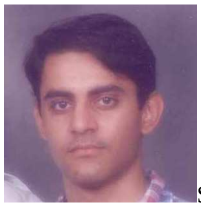
Most of the results we report are for `Capo 8.6` which somewhat outperforms `Capo 8.0` from 2000 [13] but may run slower.[7] A small number of overlaps is possible after `Capo 8.0`, therefore the authors of [13] run a commercial placer in a fast ECO mode to fix overlaps before routing. Later versions have a fast greedy built-in overlap remover and a simple detail placer based on optimal placement of small groups of cells [14]. `Capo` does not use Simulated Annealing at any point, but it is randomized. The best of five independent runs is often significantly better than the average [10]. The executable used for benchmarking was called `MetaPlacerTest0.exe`.

Dragon [61, 62, 63] performs recursive min-cut partitioning using `hMetis` libraries [37] and periodically improves global wirelength using Simulated Annealing. In our experiments `Dragon` sometimes achieves better wirelength than `Capo`, but may be an order or magnitude slower. In default mode, `Dragon` packs cells in rows left to right, which practically makes it a variable-die placer. In 2002, `Dragon` was extended with a congestion-driven mode [62] that distributes whitespace unevenly to mitigate congestion at the price of larger wirelength. `Dragon` has been tested and tuned on IBM-Place benchmarks in the same tool flow that was used to evaluate `Capo 8.0` [13]. Figure 1 shows that congestion-driven mode of `Dragon` increases wirelength compared to the normal mode. `Dragon` supports a subset of LEF/DEF. Since the source code is not available, we downloaded `Dragon` 2.20 binaries in the Fall 2002. The latest version 2.23 is primarily a bug-fix release. Most recently, timing-driven `Dragon` has been released [63].

FengShui 2.0 [65, 3] uses `hMetis` libraries [37] for recursive min-cut partitioning and attempts to further improve wirelength by using a native multi-way partitioner. `FengShui` is a variable-die placer and always packs cells in rows to the left. No data were published for `Feng Shui` describing routability with respect to major commercial routers. `Feng Shui 1.2` reported at [65] is available in source code, and `Feng Shui 2.0` is available in executable form (May 2003). We use version `2.0` unless indicated otherwise.

mPL 2.0 [24, 20] is a new multi-level placer that, at the top level, uses a fairly expensive analytical optimization [17] that directly handles non-overlapping constraints. At lower levels `mPL` uses slot assignment and enumerates permutations of small subsets of cells [30]. At the end, cells are packed to the left by sorting their locations (this is typical of a variable-die placer). `mPL 2.0` also integrates detail placement. Unlike in this paper, `mPL` wirelength is sometimes reported after the external detailed placer `Domino` [26] is applied. We noticed that those versions of `mPL` are deterministic and always produce the same placement if input is unchanged. `mPL 1.2` and, later, `mPL 1.2b, mPL 2.0` binaries for Sun/Solaris were provided to us by the authors. We ran them on an 750MHz Sparc Ultra-III processor, whereas all other placers were ran on a 2GHz Pentium4-Xeon running Linux.

[7] `Capo` can be ran in faster modes [13] via command-line parameters. However, we only report results for the default configuration.

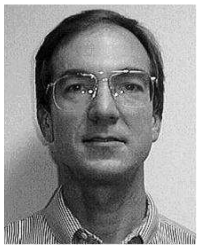
Saurabh N. Adya (S'00) received the B.E. degree in Electronics and Communication Engineering from Karnataka Regional Engineering College, Surathkal, Mangalore University, India, in 1999, and the M.S. degree in Computer Science and Engineering from University of Michigan, Ann Arbor, in 2002. He is currently working towards the Ph.D degree at the University of Michigan, Ann Arbor. From 1999 to 2000, he worked as an IC Design Engineer at Texas Instruments, India. His current research interests are in the general area of VLSI CAD and specifically, physical design for VLSI.

Mehmet C. Yildiz received the B.S. degree in Computer Engineering from Marmara University, Istanbul, Turkey in 1995, the M.S. degree in Information and Computer Science from Yeditepe University, Istanbul, Turkey in 1998 and the Ph.D degree in Computer Science from Binghamton University, New York in 2003. He is currently working at IBM Austin Research Lab as postdoc. He's main research interest is VLSI/CAD Physical Design algorithms. He's an active member of ACM SIGDA.

Igor L. Markov(M'97) received the M.A. and Ph.D degrees from the University of California, Los Angeles (UCLA), in 1994 and 2000 rescpectively.

Currently, he is an Assistant Professor in the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor. His research interests include quantum computing and combinatorial optimization with applications to the design and verification of integrated circuits. He is coauthor of more than 55 publications.

Dr. Markov is serving on the technical program committees for the International Conference on Computer-Aided Design (ICCAD), Design, Automation and Test in Europe (DATE), International Symposium on Physical Design (ISPD), Great Lakes Symposium on VLSI (GLSVLSI), International Workshop on System-Level Interconnect Prediction (SLIP) and international Workshop on Logic and Synthesis (IWLS) in 2003. He received the Best Ph.D Student Award for 2000 from UCLA.



Paul Villarrubia received a Bachelor of Science degree in Electrical Engineering in 1981 at Louisiana State University, and a Masters of Science degree from the University of Texas at Austin in 1988. He is currently a Senior Technical Staff Member at IBM where he leads the development of placement and timing closure tools. He has worked at IBM in the areas of physical design of microprocessors, physical design tools development, and tools development for ASIC timing closure. Interests include placement, synthesis, buffering, signal integrity and extraction. He won a best paper award at the 2001 ACM/IEEE Design Automation Conference.

Phiroze N. Parakh received a M.S. and Ph.D. in Electrical Engineering from the University of Michigan, Ann Arbor in 1997 and 1999 respectively. His research interests include graph partitioning, placement algorithms, and physical synthesis in the VLSI domain.

Patrick H. Madden (S'93-M'98) received the B.S. and M.S. degrees from New Mexico Tech in 1987 and 1989, respectively, and a Ph.D. from UCLA in 1998. He is an Assistant Professor of Computer Science at SUNY Binghamton. Prior to joining Binghamton, he worked for several oil field automation companies and a video games maker. His research interests include VLSI physical design and cryptography.