Optimality and Scalability Study of Existing Placement Algorithms

Chin-Chih Chang[1], Jason Cong, Min Xie

Department of Computer Science

University of California at Los Angeles

Los Angeles, CA 90095

Email: { cchang, cong, xie } @cs.ucla.edu

Abstract - Placement is an important step in the overall IC design process in DSM technologies, as it defines the on-chip interconnects, which have become the bottleneck in determining circuit performance. The rapidly increasing design complexity, combined with the demand for the capability of handling nearly flattened designs for physical hierarchy generation, poses significant challenges to existing placement algorithms. There are very few studies on understanding the optimality and scalability of placement algorithms, due to the limited sizes of existing benchmarks and limited knowledge of optimal solutions. The contribution of this paper includes two parts: 1) We implemented an algorithm for generating synthetic benchmarks that have known optimal wirelengths and can match any given net distribution vector. 2) Using benchmarks of 10K to 2M placeable modules with known optimal solutions, we studied the optimality and scalability of three state-of-the-art placers, Dragon [4], Capo [1], mPL [24] from academia, and one leading edge industrial placer, QPlace [5] from Cadence. For the first time our study reveals the gap between the results produced by these tools versus true optimal solutions. The wirelengths produced by these tools are 1.66 to 2.53 times the optimal in the worst cases, and are 1.46 to 2.38 times the optimal on the average. As for scalability, the average solution quality of each tool deteriorates by an additional 4% to 25% when the problem size increases by a factor of 10. These results indicate significant room for improvement in existing placement algorithms.

1. Introduction

Placement is an important step in the overall IC design process in DSM technologies, as it defines the on-chip interconnects, which have become the bottleneck in determining circuit performance. Existing placement algorithms can be classified into three categories, min-cut based methods [1], analytical methods [2] and iterative methods [3]. There are also many hybrid methods [4]. After producing an initial solution with one algorithm, they shift to another to further improve the solution quality.

According to ITRS'01 Roadmap [6], the maximum number of transistors per chip will be over 1.6 billion, with a clock frequency of 28.7 GHz by the year 2016. Such high complexity poses significant challenges to the scalability of placement algorithms. The traditional way to handle large designs is through partitioning according to the logical hierarchy. However, it is pointed out in [7] that these hierarchies are derived with little or no consideration for the physical layout and they may not embed well in a two-dimensional silicon surface. Therefore, it is proposed in [7] that the right way to partition the design is to first flatten the logic hierarchy to the extent that we are certain about the "physical locality" of each module in the flattened design, and then construct a physical hierarchy (coarse placement) on this almost flattened netlist. The algorithm presented in [8] is developed to support this methodology. In general, this approach requires highly scalable placement algorithms which can handle nearly flattened designs with l00K to 10M placeable objects.

Until now, there have been few studies to understand the optimality and scalability of placement algorithms. This is due to the limited sizes of existing benchmarks and limited knowledge of their optimal solutions. Two types of benchmarks are commonly used. One type of benchmarks is based on real designs [9][10][11]. They are either directly extracted from real designs [9], or based on minor perturbations of real designs [10][11]. Another type of benchmarks is synthetic benchmarks, i.e., circuits generated by computer programs. Several algorithms [13][15][16][17] can generate benchmarks with the user-specified Rent's parameter [12]. Other possible inputs to the generation algorithms include design size, net distribution vector, and logic functionality, etc. As an application of synthetic benchmarks, [18] used benchmarks from [15] to search Rent's parameter that incured the highest resource utilization ratio. The study in [19] attempted to quantify the suboptimality of placement algorithms in terms of chip area by "stitching" small designs to form large designs. The major drawback shared by these benchmarks is that their optimal solutions for placement are unknown. It is difficult to determine how the solution quality changes as the design size grows.

The contribution of this paper includes two parts: (1) We implemented an algorithm for generating synthetic benchmarks that have known optimal wirelengths and can match any given net distribution vector. Our algorithm is similar to the one first proposed by Boese, which was outlined in [19]. Boese, however, never implemented his idea nor experimented it with any placer [22]. (2) Using benchmarks of 10K to 2M placeable modules with known optimal solutions, we experimented with three state-of-the-art placers from academia, Dragon [4], Capo [1], mPL [24], and one leading edge industrial placer, QPlace [5] from Cadence.

[1] The current contacting address for Chin-Chih Chang is Cadence Design Systems Inc., 555 River Oaks Parkway, San Joese, CA 95134.

For the first time our study reveals the gap between the results produced by these tools versus true optimal solutions. The wirelengths produced by these tools are 1.66 to 2.53 times the optimal in the worst cases, and are 1.46 to 2.38 times the optimal on the average. As for scalability, the average solution quality of each tool deteriorates by an additional 4% to 25% when the problem size increases by a factor of 10. These results indicate significant room for improvement in existing placement algorithms.

The rest of this paper is organized as follows. Section 2 describes our benchmark generation algorithm. Section 3 gives experimental results using the synthetic benchmarks. Section 4 gives the conclusion and future work.

2. Placement Benchmark Generation with Known Optimal Wirelength

2.1 Problem Formulation

First, we introduce some notations: Given a netlist *N,* let *p* be the number of placeable modules in the netlist, and let $D(N) = (d_2, d_3, ...d_n)$ be the *Net Distribution Vector* (*NDV*), where d_k is the total number of *k* pin nets in the netlist.

We are interested in the following problem: Given a number *p* and a vector *D*, construct a placement benchmark with *p* placeable modules, such that its netlist has *D* as its *NDV* and has a known optimal half perimeter wirelength.

2.2 Placement Benchmark Construction Algorithm

A. Algorithm Description

Our algorithm, PEKO (Placement Example with Known Optimal wirelength), makes two assumptions: all the modules are of equal size, and there is no space between the rows. It first places all the modules in a rectangular region close to a square, then connects the nets to the modules one-by-one, using the minimum perimeter bounding box for each net. In the end, a netlist is extracted from this placed configuration. Fig.1 gives a description of the algorithm.

Fig. 2 shows an example when $p = 9$, $D = (6,2,2)$. Net A is a 4-pin net. According to our algorithm, it will connect four modules located in a 2x2 rectangular region. In Fig. 2,

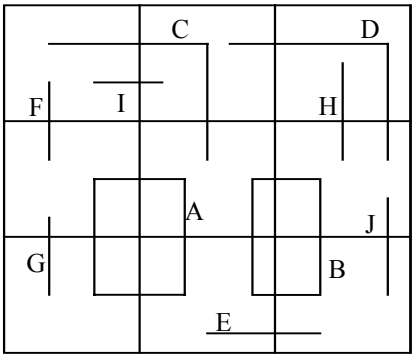

Fig. 2. Benchmark generation for $p = 9$, $D = (6,2,2)$

Algorithm PEKO

Input Total number of modules *p*
Net Distribution Vector $D = (d_2, d_3, ... d_n)$

Output Placement Benchmark having *D* as its *NDV* with known optimal wirelength

Put all the *p* modules in a $\lceil\sqrt{p}\rceil \times \lceil p/\lceil\sqrt{p}\rceil\rceil$ shaped region
$j \leftarrow 0$
For $i \leftarrow n$ to 2 **do**
  While $d_i > 0$ **do**
    Randomly pick a module *m* which has the least number of nets connected
    Randomly select a bounding box *b* that includes *m* and has size $\lceil\sqrt{i}\rceil \times \lceil i/\lceil\sqrt{i}\rceil\rceil$ (or $\lceil i/\lceil\sqrt{i}\rceil\rceil \times \lceil\sqrt{i}\rceil$)
    Generate net_j connecting *i* modules in *b*
    $d_i \leftarrow d_i$-1
    $j \leftarrow j$+1
  End While
End For
Output design size and dimension
For $i \leftarrow 0$ to j-1 **do**
  Output the modules connected by net_i
End For

Fig. 1. Algorithm for placement benchmark generation

it connects the four modules in the lower left corner. The other 4-pin net, B, is placed on the lower right corner. Similarly, the two 3-pin nets are generated as C and D respectively. This process is repeated until the *NDV* is exhausted. The total wirelength for this benchmark is 6*1+2*2+2*2 = 14.

B. Proof of Optimality

According to the generation algorithm, the wire length of each *n*-pin net is $\lceil\sqrt{n}\rceil + \lceil n/\lceil\sqrt{n}\rceil\rceil - 2$. For any *n*-pin net, the optimal half perimeter wire length can only be achieved when the modules of this net are placed in a rectangular region close to a square, i.e., the length of each side is close to $\lceil\sqrt{n}\rceil$. In particular, the width and height of the rectangle should be $\lceil\sqrt{n}\rceil$ and $\lceil n/\lceil\sqrt{n}\rceil\rceil$ respectively (or $\lceil n/\lceil\sqrt{n}\rceil\rceil$ and $\lceil\sqrt{n}\rceil$). The wirelength of such a configuration is $\lceil\sqrt{n}\rceil + \lceil n/\lceil\sqrt{n}\rceil\rceil - 2$. The wirelength of an *n*-pin net achieved by our algorithm is optimal, and the total wirelength is the sum of all the nets, therefore, it is also optimal.

Given a benchmark *E* generated by PEKO with *NDV D*, $\sum_{i=2}^{n} d_i * (\lceil\sqrt{i}\rceil + \lceil i/\lceil\sqrt{i}\rceil\rceil - 2)$ is the optimal wirelength of the benchmark, denoted as *OW*(*E*). Given a placement solution *s* to benchmark *E*, we measure its wirelength and denote it as $PW_s(E)$. We define the ratio $PW_s(E)/OW(E)$ as the *Quality Ratio* of placement solutions. This metric gives us an objective evaluation of a solution.

2.3 Generation of Realistic Benchmark Set with Known Optimal Wirelength

In order to generate realistic benchmarks, we first extract the module numbers and *NDV*s from the netlists in the ISPD98 suite [9] (originally from IBM) and generate a set of benchmarks named suite-1 using PEKO. Table 1 gives the characteristics of suite-1. The column "OW" gives the optimal half-perimeter wirelength for each benchmark. Suite-2 is generated by scaling the module number and *NDV* of each circuit in suite-1 by a factor of 10.

Table 1. Characteristics of suite-1 (suite-2 is generated by scaling the module number and *NDV* of each circuit in suite-1 by a factor of 10)

circuit	#cell	#net	#row	OW
Peko01	12506	13865	113	8.14E+05
Peko02	19342	19325	140	1.26E+06
Peko03	22853	27118	152	1.50E+06
Peko04	27220	31683	166	1.75E+06
Peko05	28146	27777	169	1.91E+06
Peko06	32332	34660	181	2.06E+06
Peko07	45639	47830	215	2.88E+06
Peko08	51023	50227	227	3.14E+06
Peko09	53110	60617	231	3.64E+06
Peko10	68685	74452	263	4.73E+06
Peko11	70152	81048	266	4.71E+06
Peko12	70439	76603	266	5.00E+06
Peko13	83709	99176	290	5.87E+06
Peko14	147088	152255	385	9.01E+06
Peko15	161187	186225	402	1.15E+07
Peko16	182980	189544	429	1.25E+07
Peko17	184752	188838	431	1.34E+07
Peko18	210341	201648	460	1.32E+07

Table 2. Characteristics of suite-3 (suite-4 is generated by scaling the module number and *NDV* of each circuit in suite-3 by a factor of 10)

circuit	#cell	#net	#row	OW
Peko01	12506	14111	113	8.22E+05
Peko02	19342	19584	140	1.27E+06
Peko03	22853	27401	152	1.51E+06
Peko04	27220	31970	166	1.76E+06
Peko05	28146	28446	169	1.95E+06
Peko06	32332	34826	181	2.07E+06
Peko07	45639	48117	215	2.89E+06
Peko08	51023	50513	227	3.15E+06
Peko09	53110	60902	231	3.65E+06
Peko10	68685	75196	263	4.75E+06
Peko11	70152	81454	266	4.72E+06
Peko12	70439	77240	266	5.02E+06
Peko13	83709	99666	290	5.89E+06
Peko14	147088	152772	385	9.03E+06
Peko15	161187	186608	402	1.16E+07
Peko16	182980	190048	429	1.25E+07
Peko17	184752	189581	431	1.35E+07
Peko18	210341	201920	460	1.32E+07

One important feature of suite-1 and suite-2 is that there is no net connected with pads. This feature is enforced from the concern that such nets may give hint about where to place each net. To make our study complete, we also generate another two sets of benchmarks which have nets connected with pads. They are named suite-3 and suite-4 respectively. Table 2 gives a description of suite-3. All benchmarks used in this paper are given in both GSRC BookShelf format and LEF/DEF format, and can be downloaded from:
http://cadlab.cs.ucla.edu/~pubbench/peko.htm.

2.4 White Space Generation

To mimic real designs, we take a simplistic approach to generate white space in the PEKO suite. After the optimal configuration is obtained, white space is inserted to the right of the placeable modules. For each circuit in PEKO, 15% of the chip area is white space.

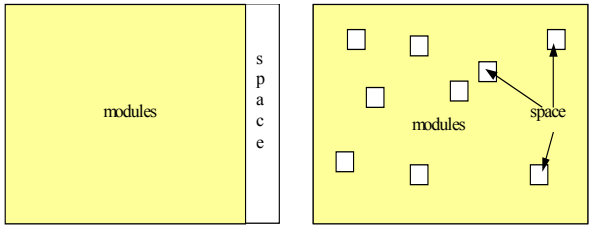

Fig. 3. White space generation methods

An alternative is to first connect each module with at least one net, then randomly remove $\alpha \times p$ modules and all the nets connected with them, where α is the ratio of desired space area to the chip area. It is easy to prove that benchmarks thus generated also have a known optimal wirelength. Furthermore, the white space is randomly distributed on the chip. This method, however, may not give a benchmark matching the desired *NDV*. Therefore, it is not used in PEKO.

3. Experimental Results and Analysis

The benchmarks are experimented with three state-of-the-art placers from academia and one leading edge industrial placer, including:

- **Dragon**: Dragon is based on a multilevel framework. It uses hMetis [21] to derive an initial partition result on the circuit, then undergoes a series of refinement stages doing bin-based swapping with simulated annealing [4]. We used Dragon v.2.20 downloaded from [25].
- **Capo**: Capo is built on a multilevel partitioner. It aims to enhance the routability with such techniques as tolerance computation and block splitting heuristics [1]. We used Capo v.8.0 downloaded from [26].
- **mPL**: mPL is also based on a multilevel framework. It uses nonlinear programming to handle the non-overlapping constraints on the coarsest level, then uses Goto [23] based relaxation in subsequent refinement stages [20]. We used mPL v.1.2 in our experiment. Compared with [20], mPL v.1.2 uses an additional V cycle and does distance-based clustering in the second V cycle [24].
- **QPlace**: QPlace [5] is the placement engine used in the Silicon Ensemble of Cadence. The version we used is QPlace v.5.1.55 in Silicon Ensemble v.5.3.

Experiments with Dragon, QPlace, and mPL are performed on a SUN Blade 750 MHz running SunOs 5.8 with 4GB of memory. Capo's binary is not compatible with our UNIX system, therefore the experiments with Capo are performed on a Pentium IV 1.4GHz running Windows 2000 with 2GB of memory. The data is collected after running each tool only once. Since all the tools make use of randomization, running them several times may give different results. Also, direct

comparison of Capo's runtime with the other tools may not be meaningful as it is run on a different machine, but the runtime data can give us some idea about its speed and scalability. We need to emphasize that it is not our purpose to give a comparison of the four placers. The experiments are performed to determine how much room is left for improvement in existing placement algorithms.

The test results for suite-1 are given in Table 3. The column "PW" gives the detail placement wirelength produced by each tool. For each benchmark, the Quality Ratio is calculated for the four tools and given in the columns named "QR." According to the experiments, none of these tools achieve a Quality Ratio close to 1. The wirelengths produced by these tools can be 1.66 to 2.53 times the optimal in the worst cases[2]. It may be possible that some of the placers will try to enhance the routability by sacrificing the wirelength. However, given the gap between their wirelengths and the optimal value, there remains significant room for improvement in existing placement algorithms.

The entire test is repeated on suite-2 to observe how the QRs change as the design size grows. Since the benchmark sizes are 10x larger in this set, we set an upper limit of 24 hours to a tool's runtime. The results are given in Table 4. QPlace scales well in terms of runtime. It finishes 16 out of 18 benchmarks (up to 1.83M placable modules), and runs out of memory on the remaining two (with 1.85M and 2.15M placeable modules) on our machine's configuration. Its average Quality Ratio increases by only 4% from 1.84 to 1.88. Of the four tools, this increase is the smallest. Capo also shows good scalability in runtime. It finishes 13 of the circuits (up to 837K placeable modules) and runs out of memory on the remaining 5 circuits. Its average Quality Ratio shows an increase of 16% with the increase in design size. mPL finishes 9 of the 18 benchmarks, and runs out of memory on the remaining circuits. Its average Quality Ratio increases by 25% from 1.46 to 1.71. This is the highest increase of the four placers. Dragon manages to complete the placement for only the first 6 benchmarks (up to 323K placable modules) within 24 hours. Its average Quality Ratio increases from 2.09 to 2.28.

Fig. 4 and Fig. 5 give the combined results for suite-1 and suite-2. They show how the solution quality and runtime of each tool change with the increase in cell numbers.

Table 5 and Table 6 give the experimental results for suite-3 and suite-4, which have nets connected with pads. For these circuits, the wirelengths produced by the placers are 1.53 to 2.49 times the optimal in the worst cases, and are 1.43 to 2.37 times the optimal on the average. Their average solution quality shows deterioration by an additional 5% to 21% when the problem size increases by a factor of 10.

It can be seen from Table 3 to Table 6 that having nets connected with pads does give some hint about the optimal solution to some placers, with 12% improvement by mPL and 10% improvement by QPlace. This is understandable as analytical placement algorithms make use of fixed pad locations to avoid de-generated solutions. Interesting enough, Capo and Dragon do not benefit from the additional information from connection to pads.

Although our algorithm is capable of generating arbitrarily-sized benchmarks with known optimal wirelength, given the scalability problems encountered by these tools on suite-2 and suite-4, it is not meaningful for us to construct larger designs to further evaluate these algorithms.

[2] We were provided with Capo's latest version by its author when preparing the final version of this paper. Some preliminary experiments did show improvement in solution quality. However, due to the time limitation, we could not finish all the experiments and include the results here.

4. Conclusion and Future work

In this paper, we implemented an algorithm to generate synthetic placement benchmarks with known optimal wirelength matching any net distribution vector. Using benchmarks of 10K to 2M placeable modules with known optimal solutions, we experimented with four state-of-the-art placement tools. The wirelengths produced by these tools are 1.66 to 2.53 times the optimal in the worst cases, and are 1.46 to 2.38 times the optimal on the average. As for scalability, the average solution quality of each tool shows deterioration by an additional 4% to 25% when the problem size increases by a factor of 10.

Our study, as reported in this paper, is by no means complete. We did not have a chance to experiment with a number of well known placers, such as Gordian-L [2], TimberWolf [3], mPG [8], from academia, and placement engines used by Avant!, Magma, and Synopsys. Also, the benchmarks generated by our algorithm have several limitations. For example, all modules in these circuits are of uniform size, making them unsuitable for evaluating the legalization capability of detail placement algorithms. All the nets in the optimal solutions are local connections, i.e., they only connect modules in contiguous areas. This may not be true in real circuits. Therefore, obtaining good results for these benchmarks may not guarantee good solution quality in real circuits. Also, these benchmarks can not be used to evaluate routability and delay. Nevertheless, we have made a very important step in understanding the optimality and scalability of existing placement algorithms. We plan to further enhance our benchmark construction algorithm and broaden its applicability in the future.

Acknowledgements

This work is partially supported by Semiconductor Research Corporation under Contract 98-TJ-686, and by National Science Foundation under Grant CCR0096383. The authors would like to thank X. Yuan for the valuable discussions with her. They would also like to thank J. Shinnerl and K. Sze for providing the experimental data of mPL v.1.2.

Table 3. Experimental results for suite-1

		Dragon			QPlace			Capo			mPL		
circuit	OW	PW	QR	runtime(s)	PW	QR	runtime(s)	PW	QR	runtime(s)	PW	QR	runtime(s)
Peko01	8.14E+05	1.46E+06	1.79	1062	1.38E+06	1.69	91	1.83E+06	2.24	37	1.14E+06	1.40	88
Peko02	1.26E+06	2.43E+06	1.93	1823	2.29E+06	1.82	124	2.80E+06	2.22	62	1.86E+06	1.48	159
Peko03	1.50E+06	2.93E+06	1.95	2037	2.85E+06	1.90	145	3.42E+06	2.28	78	2.20E+06	1.47	185
Peko04	1.75E+06	3.87E+06	2.21	3839	3.33E+06	1.90	172	4.00E+06	2.28	94	2.33E+06	1.33	193
Peko05	1.91E+06	3.79E+06	1.98	5754	3.72E+06	1.95	242	4.33E+06	2.27	102	2.76E+06	1.45	250
Peko06	2.06E+06	4.35E+06	2.11	4599	3.60E+06	1.75	261	4.84E+06	2.35	116	2.94E+06	1.43	243
Peko07	2.88E+06	6.24E+06	2.17	3268	5.31E+06	1.85	349	7.04E+06	2.44	172	4.29E+06	1.49	342
Peko08	3.14E+06	6.79E+06	2.16	8353	5.60E+06	1.78	435	7.13E+06	2.27	198	4.38E+06	1.39	465
Peko09	3.64E+06	7.72E+06	2.12	6930	6.93E+06	1.91	475	8.64E+06	2.38	220	4.89E+06	1.34	426
Peko10	4.73E+06	8.49E+06	1.79	9208	8.89E+06	1.88	593	1.14E+07	2.42	301	7.18E+06	1.52	590
Peko11	4.71E+06	9.11E+06	1.94	7672	9.24E+06	1.96	560	1.12E+07	2.38	297	6.69E+06	1.42	525
Peko12	5.00E+06	9.67E+06	1.93	10411	9.01E+06	1.80	684	1.25E+07	2.50	324	6.83E+06	1.37	619
Peko13	5.87E+06	1.26E+07	2.15	10104	1.04E+07	1.78	742	1.39E+07	2.36	387	8.40E+06	1.43	642
Peko14	9.01E+06	1.81E+07	2.01	13354	1.59E+07	1.76	1320	2.18E+07	2.42	691	1.45E+07	1.61	1161
Peko15	1.15E+07	2.51E+07	2.17	18133	2.14E+07	1.85	1719	2.77E+07	2.40	890	1.61E+07	1.39	1484
Peko16	1.25E+07	2.81E+07	2.26	20111	2.29E+07	1.83	1928	3.11E+07	2.50	998	2.07E+07	1.66	1685
Peko17	1.34E+07	3.36E+07	2.50	40440	2.52E+07	1.87	2126	3.40E+07	2.53	1063	1.94E+07	1.44	1900
Peko18	1.32E+07	3.15E+07	2.38	36988	2.42E+07	1.84	2074	3.33E+07	2.53	1113	2.14E+07	1.62	2015
Avg.			2.09			1.84			2.38			1.46	

Table 4. Experimental results for suite-2

		Dragon			QPlace			Capo			mPL		
circuit	OW	PW	QR	runtime(s)	PW	QR	runtime(s)	PW	QR	runtime(s)	PW	QR	runtime(s)
Peko01x10	8.14E+06	1.75E+07	2.15	22197	1.44E+07	1.77	1186	1.95E+07	2.39	616	1.48E+07	1.82	1068
Peko02x10	1.26E+07	3.19E+07	2.53	31093	2.45E+07	1.94	2106	3.28E+07	2.61	1054	1.99E+07	1.58	1762
Peko03x10	1.50E+07	3.79E+07	2.53	32316	2.75E+07	1.83	2397	3.66E+07	2.44	1314	2.57E+07	1.71	2110
Peko04x10	1.75E+07	3.60E+07	2.05	34125	3.25E+07	1.85	2736	4.35E+07	2.48	1592	2.80E+07	1.60	2397
Peko05x10	1.91E+07	3.85E+07	2.02	52703	3.51E+07	1.84	3126	4.82E+07	2.53	1752	2.93E+07	1.54	2987
Peko06x10	2.06E+07	4.99E+07	2.42	48287	3.98E+07	1.93	3214	5.06E+07	2.45	1981	3.35E+07	1.62	3013
Peko07x10	2.88E+07	NA	NA	>24h	5.32E+07	1.85	4797	7.43E+07	2.58	3074	5.44E+07	1.89	4740
Peko08x10	3.14E+07	NA	NA	>24h	5.97E+07	1.90	6055	7.59E+07	2.42	3591	5.68E+07	1.81	6414
Peko09x10	3.64E+07	NA	NA	>24h	6.71E+07	1.84	6127	9.41E+07	2.59	4017	6.59E+07	1.81	5574
Peko10x10	4.73E+07	NA	NA	>24h	8.99E+07	1.90	7600	1.24E+08	2.62	5783	NA	NA	out of mem
Peko11x10	4.71E+07	NA	NA	>24h	9.10E+07	1.93	7586	1.22E+08	2.58	5835	NA	NA	out of mem
Peko12x10	5.00E+07	NA	NA	>24h	9.10E+07	1.82	8888	1.31E+08	2.63	6184	NA	NA	out of mem
Peko13x10	5.87E+07	NA	NA	>24h	1.09E+08	1.86	9905	1.57E+08	2.67	7861	NA	NA	out of mem
Peko14x10	9.01E+07	NA	NA	>24h	1.74E+08	1.93	17757	NA	NA	out of mem	NA	NA	out of mem
Peko15x10	1.15E+08	NA	NA	>24h	2.26E+08	1.96	23054	NA	NA	out of mem	NA	NA	out of mem
Peko16x10	1.25E+08	NA	NA	>24h	2.33E+08	1.87	26903	NA	NA	out of mem	NA	NA	out of mem
Peko17x10	1.34E+08	NA	NA	>24h	NA	NA	out of mem	NA	NA	out of mem	NA	NA	out of mem
Peko18x10	1.32E+08	NA	NA	>24h	NA	NA	out of mem	NA	NA	out of mem	NA	NA	out of mem
Avg.			2.28			1.88			2.54			1.71	

Table 5. Experimental results for suite-3

		Dragon			QPlace			Capo			mPL		
circuit	OW	PW	QR	runtime(s)	PW	QR	runtime(s)	PW	QR	runtime(s)	PW	QR	runtime(s)
Peko01	8.22E+05	1.67E+06	2.03	1151	1.41E+06	1.71	88	1.83E+06	2.23	39	1.14E+06	1.39	94
Peko02	1.27E+06	2.63E+06	2.08	1939	2.13E+06	1.68	147	2.89E+06	2.28	64	1.88E+06	1.48	171
Peko03	1.51E+06	2.95E+06	1.95	2127	2.77E+06	1.84	179	3.41E+06	2.26	78	2.21E+06	1.46	201
Peko04	1.76E+06	3.88E+06	2.20	4046	3.03E+06	1.72	211	3.95E+06	2.24	95	2.37E+06	1.35	199
Peko05	1.95E+06	4.14E+06	2.13	6051	3.22E+06	1.65	220	4.41E+06	2.26	104	2.95E+06	1.52	257
Peko06	2.07E+06	4.21E+06	2.03	5106	3.47E+06	1.68	271	4.81E+06	2.33	116	2.89E+06	1.40	246
Peko07	2.89E+06	6.46E+06	2.24	3560	5.11E+06	1.77	370	7.13E+06	2.47	175	4.20E+06	1.45	354
Peko08	3.15E+06	6.60E+06	2.10	9257	5.44E+06	1.73	493	7.24E+06	2.30	199	4.36E+06	1.38	468
Peko09	3.65E+06	7.37E+06	2.02	7552	6.13E+06	1.68	517	8.85E+06	2.43	218	4.90E+06	1.34	447
Peko10	4.75E+06	9.00E+06	1.89	10155	8.89E+06	1.87	663	1.14E+07	2.40	305	7.25E+06	1.53	578
Peko11	4.72E+06	8.83E+06	1.87	8026	8.22E+06	1.74	633	1.09E+07	2.31	303	6.60E+06	1.40	515
Peko12	5.02E+06	9.89E+06	1.97	11086	8.70E+06	1.73	713	1.25E+07	2.49	327	7.03E+06	1.40	618
Peko13	5.89E+06	1.19E+07	2.03	10431	1.06E+07	1.80	759	1.41E+07	2.40	389	8.34E+06	1.42	680
Peko14	9.03E+06	1.83E+07	2.03	13011	1.55E+07	1.71	1357	2.22E+07	2.46	702	1.38E+07	1.53	1212
Peko15	1.16E+07	2.58E+07	2.23	17189	1.99E+07	1.73	1875	2.87E+07	2.48	907	1.55E+07	1.34	1555
Peko16	1.25E+07	2.83E+07	2.26	18874	2.23E+07	1.79	1970	3.06E+07	2.45	1090	1.80E+07	1.44	1749
Peko17	1.35E+07	3.18E+07	2.36	37719	2.29E+07	1.70	2272	3.35E+07	2.49	1084	1.94E+07	1.44	1913
Peko18	1.32E+07	3.24E+07	2.45	33537	2.26E+07	1.71	2293	3.19E+07	2.42	1140	1.93E+07	1.46	2115
Avg.			2.10			1.74			2.37			1.43	

Table 6. Experimental results for suite-4

		Dragon			QPlace			Capo			mPL		
circuit	OW	PW	QR	runtime(s)	PW	QR	runtime(s)	PW	QR	runtime(s)	PW	QR	runtime(s)
Peko01x10	8.22E+06	1.85E+07	2.25	24149	1.40E+07	1.71	1075	2.03E+07	2.47	623	1.28E+07	1.56	1131
Peko02x10	1.27E+07	3.16E+07	2.49	33959	2.20E+07	1.73	1700	3.24E+07	2.56	1078	1.99E+07	1.57	1816
Peko03x10	1.51E+07	3.76E+07	2.49	34191	2.65E+07	1.76	1959	3.72E+07	2.46	1341	2.45E+07	1.62	2169
Peko04x10	1.76E+07	3.56E+07	2.02	36569	3.10E+07	1.76	2266	4.39E+07	2.49	1628	2.72E+07	1.54	2441
Peko05x10	1.95E+07	4.17E+07	2.14	57099	3.46E+07	1.78	2657	4.87E+07	2.50	1808	3.67E+07	1.89	3214
Peko06x10	2.07E+07	5.12E+07	2.48	52540	3.71E+07	1.79	3042	5.54E+07	2.68	2035	3.17E+07	1.53	3217
Peko07x10	2.89E+07	NA	NA	>24h	5.24E+07	1.82	4429	7.25E+07	2.51	3128	4.57E+07	1.58	4609
Peko08x10	3.15E+07	NA	NA	>24h	5.42E+07	1.72	5748	7.78E+07	2.47	3676	4.82E+07	1.53	6219
Peko09x10	3.65E+07	NA	NA	>24h	6.51E+07	1.78	5411	9.87E+07	2.71	4127	5.43E+07	1.49	5531
Peko10x10	4.75E+07	NA	NA	>24h	8.92E+07	1.88	6835	1.27E+08	2.67	5971	NA	NA	out of mem
Peko11x10	4.72E+07	NA	NA	>24h	8.71E+07	1.84	6742	1.25E+08	2.64	5918	NA	NA	out of mem
Peko12x10	5.02E+07	NA	NA	>24h	9.20E+07	1.83	7259	1.35E+08	2.69	6381	NA	NA	out of mem
Peko13x10	5.89E+07	NA	NA	>24h	1.05E+08	1.78	8384	1.55E+08	2.63	8093	NA	NA	out of mem
Peko14x10	9.03E+07	NA	NA	>24h	1.61E+08	1.78	14424	NA	NA	out of mem	NA	NA	out of mem
Peko15x10	1.16E+08	NA	NA	>24h	2.18E+08	1.88	22499	NA	NA	out of mem	NA	NA	out of mem
Peko16x10	1.25E+08	NA	NA	>24h	2.20E+08	1.76	28082	NA	NA	out of mem	NA	NA	out of mem
Peko17x10	1.35E+08	NA	NA	>24h	NA	NA	out of mem	NA	NA	out of mem	NA	NA	out of mem
Peko18x10	1.32E+08	NA	NA	>24h	NA	NA	out of mem	NA	NA	out of mem	NA	NA	out of mem
Avg.			2.31			1.79			2.58			1.59	

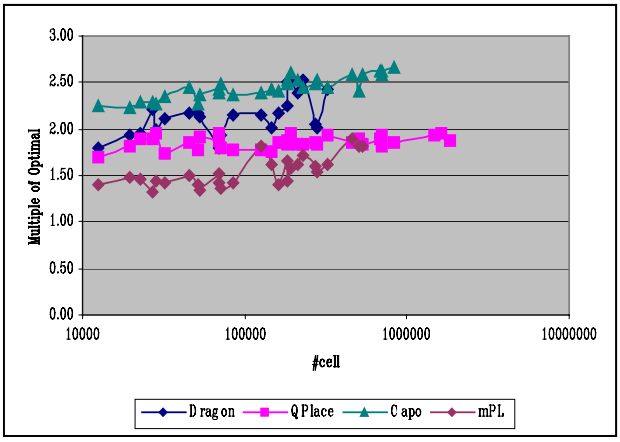

Fig. 4. Solution quality vs cell number (combining suite-1 and suite-2)

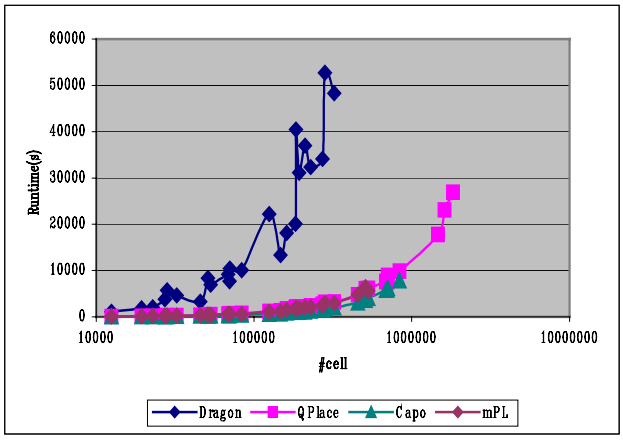

Fig. 5. Runtime vs cell number (combining suite-1 and suite-2)

References

[1] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Can Recursive Bisection Produce Routable Placements?" *Proc. Design Automation Conference*, pp. 477-482, 2000.

[2] G. Sigl, K. Doll, and F. Johannes, "Analytical placement: A linear or quadratic objective function?" *Proc. Design Automation Conference*, pages 427-432, 1991.

[3] C. Sechen and A. Sangiovanni-Vincentelli, "The TimberWolf Placement and Routing Package," *IEEE J of Solid-State Circuits* Vol. 20 No. 2, pp. 432-439, 1985.

[4] M. Wang, X. Yang and M. Sarrafzadeh, "Dragon2000: Standard-cell Placement Tool for Large Industry Circuits," *Proc. International Conference on Computer-Aided Design*, pp. 260-264, 2000.

[5] Cadence Design Systems Inc, "Envisia Ultra Placer Reference," QPlace version 5.1.55, compiled on 10/25/1999.

[6] International Technology Roadmap for Semiconductors 2001 Edition.

[7] J. Cong, "An Interconnect-Centric Design Flow for Nanometer Technologies," *Proceedings of the IEEE*, Vol. 89, No. 4, pp 505-528, 2001.

[8] C-C. Chang, J. Cong, Z. Pan, X. Yuan, "Physical Hierarchy Generation with Routing Congestion Control," *Proc. International Symposium on Physical Design,* pp. 36-41, 2002.

[9] C. J. Alpert, "The ISPD98 Circuit Benchmark Suite," *Proc. International Symposium on Physical Design*, pp. 85-90, 1998.

[10] D. Ghosh, N. Kapur, and F. Brglez. "Toward A New Benchmarking Paradigm in EDA: Ananalysis of Equivalent Class Mutant Circuit Distribution," *Proc. International Symposium on Physical Design,* pp. 136-144, 1997.

[11] K. Iwama and K. Hino, "Random Generation of Test Instance for logic Optimizers," *Proc. Design Automation Conference*, pp. 430-434, 1994.

[12] B. S. Landman and R. L. Russo. “On a pin versus block relationship for partitions of logic graphs,” *IEEE Trans. on Computer*s, C20, pp. 1469-1479, 1971.

[13] J. Darnauer and W.M. Dai. “A Method for Generating random circuits and its application to routability measurement,” *Proc. International Symposiu. on Field Programmable Gate Arrays*, pp. 66-72, 1996.

[14] P. Verplaetse, J. Van Campenhout, and D. Stroobandt, “On synthetic benchmark generation methods,” *Proc International Symposium on Circuits and Systems*, pp. 213-216, 2000.

[15] D. Stroobandt, P. Verplaetse, and J. Van Campenhout. “Generating synthetic benchmark circuits for evaluating CAD tools,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol 19 No 9, pp. 1011-1022, 2000.

[16] D. Stroobandt, P. Verplaetse, and J. Van Campenhout, “Towards synthetic benchmark circuits for evaluating timing-driven cad tools,” *Proc. International Symposium of Physical Design,* pp. 60-66. 1999.

[17] M.D. Hutton, J. Rose, J.P. Grossman, and D. Corneil, “Characterization and parameterized generation of synthetic combinatinational circuits,” *IEEE Trans. on Computer-Aided Design*, pp. 985-996, 1998.

[18] G. Parthasarthy, M. Marek-Sadowska, A. Mukherjee and A. Singh, “Interconnect Complexity-Aware FPGA Placement using Rent's rule,” *Proc. System-Level Interconnect Prediction*, 2001.

[19] Lars W. Hagen, Dennis J.-H. Huang and Andrew B. Kahng, “Quantified Suboptimality of VLSI Layout Heuristics,” *Proc. Design Automation Conference*, pp. 216-221, 1995.

[20] Tony F. Chan, Jason Cong, Tianming Kong, and Joseph R. Shinnerl, “Multilevel Optimization for Large-Scale Circuit Placement,” Proc *Proc. International Conference on Computer-Aided Design*, pp. 171-176, 2000

[21] G. Karypis, B. Aggarwal, V. Kumar and S. Shekhar, “Multi-level hypergraph partitioning: Application in VLSI domain”, *IEEE Trans. VLSI Syst, vol. 7, pp. 69-79, Mar. 1999.*

[22] K. Boese, *personal communication*, 2002.

[23] S. Goto, “An efficient algorithm for the two dimensional placement problem in electrical circuit layout,” *IEEE Trans. on Circuit and Systems*, vol 28, pp. 12-18, 1981.

[24] K. Sze, *personal communication*, 2002.

[25] http://er.cs.ucla.edu/Dragon/download.html.

[26] http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/Placement/bin/