A Trade-off Oriented Placement Tool[1]

Huaiyu Xu **Maogang Wang**[†] **Bo-Kyung Choi** **Majid Sarrafzadeh**

Computer Science Department
University of California, Los Angeles, CA, 90095
huaiyu@cs.ucla.edu, mgwang@cadence.com, bkchoi,majid@cs.ucla.edu

Abstract

High quality placement results are always produced at the cost of significant runtimes. In this paper, we study the trade-off between the overall quality and the runtime for standard-cell placement problems. We implemented and studied a class of schemes to achieve the runtime vs. quality trade-off. We developed a new trade-off oriented placement tool (TOOP) which is controlled by decision trees. TOOP can adjust itself based on user's requests and netlist properties. Compared to Cadence QPlace, even the fastest mode of TOOP (lowest quality) can produce placements with similar or better layout. TOOP also shows much stronger ability to produce routable placement when compared to Capo.

Keywords

Placement, Quality, Runtime

1 INTRODUCTION

Standard-cell placement is a fundamental problem in the VLSI physical design area. It has been drawing massive attentions in the VLSI CAD field for more than twenty years. Even the most classical placement problem (minimizing total wirelength) is still a very active topic among researchers [1, 2, 3, 4, 5, 7]. Being an NP-hard problem, the placement problem is unlikely to be solved optimally within a reasonable amount of time. On the other hand, the problem size keeps the exponentially increasing trend, which makes old placement heuristics less and less effective. In order to handle current multi-million gates design, a state-of-the-art placement tool typically consists of several heuristics with each of them focused on a special sub-problem.

There are two essential aspects to consider in a VLSI design process: quality and time to complete the design. Ideally, designers would love to have designs with the highest quality obtained within the shortest amount of time. However, as the problem itself being NP-hard, quality and time inevitably trades off each other. A good placement tool should be very flexible to adapt itself based on different requests from designers.

Almost all existing placement research works focused on improving the quality or/and speeding up the runtime over an existing algorithm. Almost none has been documented in a way of adaptively controlling the trade-off between quality and runtime. In this paper, we develop a trade-off oriented placement tool (TOOP). We seek to establish an adaptive way for our placer to perform placement based on both requests from users and the properties of the netlist. The actual placement process is controlled by decision trees.

The rest of the paper is organized as follows. Section 2 briefly describes the framework of our trade-off oriented placement tool. In section 3, several blocking schemes are introduced and the decision trees are described to control our placement tool. The experimental results are shown in section 4 by comparing our placement tool with Cadence QPlace and an academic placement tool Capo[2]. Section 5 is the conclusion.

2 FRAMEWORK OF TOOP

We use a high-quality academic placement tool, Dragon [5] as the barebones of our trade-off oriented placement system. Components and controls will be added to Dragon to achieve the desired trade-off between quality and runtime. Dragon consists of several parts including partitioning, clustering, simulated-annealing based optimization and local greedy improvement, etc. Since we use Dragon to construct our trade-off oriented placement tool, it is worthwhile to spend a paragraph here to briefly review how Dragon works.

Dragon works in a top-down fashion. At each hierarchical level, quadrisection is performed topologically on the netlist as well as physically on the layout area. At the top level, the layout area is divided into four areas and we call these areas "**bins**". Meanwhile, the netlist is partitioned into four cell **clusters** with similar size to minimize the interconnections between them. The next step is to put clusters into bins. Simulated annealing is used to minimize the total wirelength. This phase is called "**bin annealing**". These steps are recursively performed until each cluster only contains a small number of cells. The "**cell annealing**" phase is performed after bin annealing. It moves single cells around bins to further reduce wirelength. Finally, the detailed placement phase removes overlaps between cells and greedily improves the final wirelength. Our new trade-off placement tool is constructed on top of Dragon. As shown in Fig. 1, controls

[1]This work was supported by NSF under Grant #CCR-0090203

[†]Dr. Maogang Wang is with Cadence Design Systems, San Jose, CA 95134.

based on decision trees are added to the system to control the interactions between each sub-component inside Dragon.

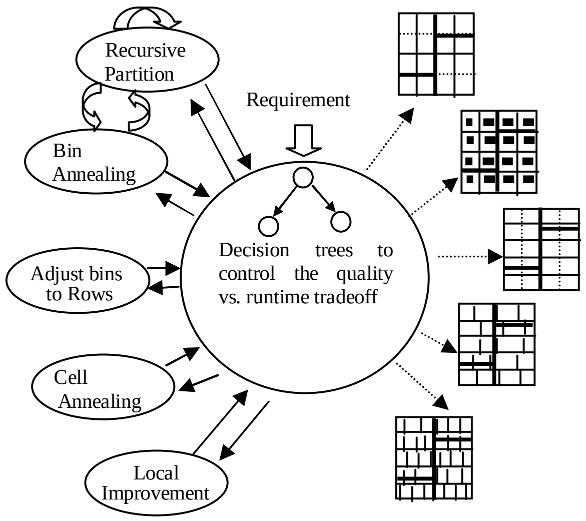


Fig. 1 Framework of our placement tool

3 DETAILED IMPLEMENTATION

In this section, we illustrate detailed schemes and controls inside our trade-off oriented placement tool.

All results shown are obtained by running experiments on a SUN Ultra10 workstation with a 400MHz CPU. TOOP is tested on selected benchmarks from the IBM placement benchmark suite. The placer reads LEF/DEF files and outputs placement results in the DEF format. The Cadence WarpRouter is used to read the placement and perform the routing. The properties of the benchmarks we used are summarized in Table 1.

Table 1. Properties of circuit used in experiments

circuits	cells	nets	rows	white space	core(row) util.	routing layers
ibm01	12,028	11,753	132	14.88%	85.12%	4
ibm07	44,811	44,681	233	10.05%	89.95%	5
ibm08	50,672	48,230	243	9.97%	90.03%	5
ibm09	51,382	50,678	246	9.76%	90.24%	5
ibm10	66,762	64,971	321	9.78%	90.22%	5
ibm11	68,046	67,422	281	9.89%	90.11%	5
ibm12	68735	68,376	347	14.78%	85.22%	5

In most top-down partitioning based placement algorithms, interactions between partitions is prohibited. Once a cell is assigned to a certain partition, it will stay there throughout the whole placement process. This helps to solve the original problem in a true divide-and-conquer manner. However, an obvious disadvantage for this approach is that a cell's position is restricted by a decision which is made early in the process. To address this issue, Dragon allows cells be moved between partitions.

We can describe this behavior by introducing the notion of **"block"**. A block contains a set of partitions and acts as a fence to restrict the moves of cells which belong to this set of partitions. No cell can be moved outside the block boundary (fence). In a classical top-down placement algorithm, a block only contains one partition. Thus a cell is always staying within this partition. For Dragon, there is only one block which contains all the partitions. Theoretically cells can be moved to anywhere within the whole layout area at each hierarchical level. Besides these two extreme cases, a block may contain an arbitrary number of partitions and the boundary of the block need not to be rectangular. Generally speaking, a large block size is connected with a better final placement but a longer runtime.

3.1 Pre-fixed Blocking Scheme

To test the first knob in our trade-off oriented placement tool, we use three pre-fixed blocking approaches (Approach A, B, C as shown in Fig. 2). At the first hierarchical level, there are 4 bins and 4 cell clusters but only one block in the entire layout area. At the second level, each bin, cluster and block will be divided into 4 smaller parts. Thus we have 4 blocks, 16 bins and 16 clusters in total. In approach A, these 4 second level blocks are obtained by crossly cutting the first level block. While in approach B and C, these 4 second level blocks are obtained by cutting the first level block vertically and horizontally, respectively. Each of these 4 blocks has 4 bins and 4 clusters inside. Four clusters belonged to the same block can only be shuffled within the block during the bin annealing phase.

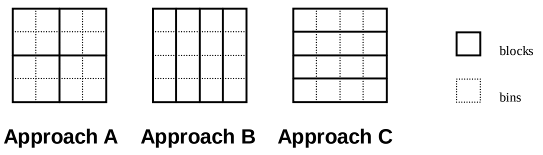


Fig. 2. Pre-fixed blocking schemes

The purpose of defining/using blocks is to restrict the freedom of cluster movement when doing bin annealing. Thus the size of the solution space for each cluster placement problem is greatly reduced. We test these three approaches on a set of benchmark circuits. The interesting fact is that we found Approach B performs the best among these three approaches. We compare the total wirelength after placement and routing of Dragon and wirelength of the pre-fixed blocking scheme. Table 2 shows the results. On average, pre-fixed blocking scheme can speed up Dragon by a factor of 2.7. On the other hand, the quality is worsening by 10%.

Table 2. Final placement comparison between Dragon and TOOP with the pre-fixed blocking scheme

circuits	placer	place. WL	*routed* WL	vios	place. time(s)	speedup	WL degra.
ibm01	dragon	0.58	0.86	0	1803		
	TOOP	0.63	0.90	0	547	3.3x	10%
ibm07	dragon	3.55	4.55	0	5002		
	TOOP	3.96	4.79	0	1998	2.5x	12%
ibm09	dragon	3.21	3.77	0	8736		
	TOOP	3.54	4.05	0	3415	2.6x	7%
ibm11	dragon	4.77	5.50	0	9953		
	TOOP	5.25	6.33	0	4380	2.3x	10%

3.2 Interconnection-Aware Blocking Scheme

The pre-fixed blocking scheme can effectively reduce the size of the solution space for each cluster placement problem. On the other hand, it blindly posts artificial regulations on clusters to restrict where they can be moved. A better way would be determining the block size and shape dynamically to minimize the interconnection between them.

When minimizing interconnections, we could have arbitrary shaped blocks. To validate this idea, we implement the interconnection-aware scheme by making two simplifications: 1). All the blocks are rectangular. 2). each block will be crossly cut (meaning one horizontal cut followed by one vertical cut) to get smaller blocks at the next hierarchical level. This approach is illustrated in Fig. 3. When the horizontal cut is performed, we check all possible horizontal cuts in the current block and pick the one which results in the fewest interconnections at the cut-line. The same procedure is performed to do the vertical cut.

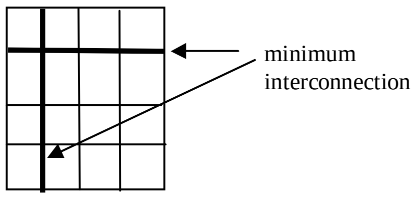


Fig. 3. Interconnection-aware blocking scheme

We compare placement results of this interconnection-aware blocking scheme with placement results of Dragon in Table 3. On average, interconnection-aware blocking scheme can speed up Dragon by a factor of 1.3 with a quality loss of 7%.

3.3 Congestion-Aware Blocking Scheme

Congestion is one of the most important metrics in modern placement problem. In some high-utilized designs, we need to especially be aware of congestion during placement. In this subsection, we propose a congestion-aware blocking scheme to help reduce location congestion during placement.

Table 3. Final placement comparison between Dragon and the interconnection-aware blocking scheme

circuits	placer	place. WL	*routed* WL	vios	place. time(s)	speedup	WL degra..
ibm01	dragon	0.58	0.86	0	1803		
	TOOP	0.61	0.88	0	1379	1.2x	5%
ibm07	dragon	3.55	4.55	0	5002		
	TOOP	3.80	4.83	0	3552	1.4x	9%
ibm09	dragon	3.21	3.77	0	8736		
	TOOP	3.37	3.94	0	7035	1.2x	6%
ibm11	dragon	4.77	5.50	0	9953		
	TOOP	5.06	5.72	0	7907	1.3x	6%

The Cheng's bounding box model is selected to evaluate congestion during placement in our placement tool [6]. To identify the congested area, we define the average congestion in a bin b(i, j) as

$$C_{ij} = \frac{1}{4}\left(\frac{C_{ij,h}}{C_{avg,h}} + \frac{C_{(i-1)j,h}}{C_{avg,h}} + \frac{C_{ij,h}}{C_{avg,v}} + \frac{C_{i(j-1),h}}{C_{avg,v}}\right)$$

Where $C_{avg,v}$ and $C_{avg,h}$ are vertical and horizontal crossings interconnections for each edge of the bin, respectively. They can be obtained by

$$C_{avg,h} = \frac{1}{(m-1)n}\sum_{i=1}^{m-1}\sum_{j=1}^{n} C_{ij,h}$$

$$C_{avg,h} = \frac{1}{(m-1)n}\sum_{i=1}^{m-1}\sum_{j=1}^{n} C_{ij,v}$$

The purpose of using the congestion-aware blocking scheme is to help placer budgeting congestion distribution well to avoid local congested spot in the final placement. We set the blocks in a way to balance bin congestion. Similar to what we did in the interconnection-aware blocking scheme, we cut the current block twice (one horizontal cut followed by one vertical cut). The criterion for each cut is to balance the average bin congestion on both sides of the cut-line.

3.4 Decision Tree Based Trade-off Oriented Placement Tool

In previous subsections, we introduced several blocking schemes (pre-fixed, interconnection-aware and congestion-aware). Each of them has their own focuses and tradeoffs. To briefly summarize, the pre-fixed blocking scheme is the fastest among these three. The interconnection-aware blocking scheme produces the best placement in terms of final wirelength and is relatively fast. The congestion-aware scheme focuses on solving the local congestion problem in the final placement, but it may lose some placement quality. Based on these properties, we can construct a "Blocking Decision Tree (BDT)" to decide which individual blocking scheme to use for each block at a hierarchical level.

As shown in Fig. 4, we start traversing BDT by looking at congestion distribution among all bins at this level. If the total amount of congestion exceed a certain threshold, we declare this block as congested and use congestion-aware blocking scheme on this block. If the current block is not congested, we look at the number of interconnections for each bin inside this block. If the total number of interconnections exceeds a certain threshold, we use interconnection-aware blocking scheme, otherwise we use the fastest pre-fixed blocking scheme. Basically, BDT tries to use the fastest blocking scheme (pre-fixed) on blocks it deems as "easy" for placer to handle and the most complicated blocking scheme (congestion-aware) on "difficult" blocks.

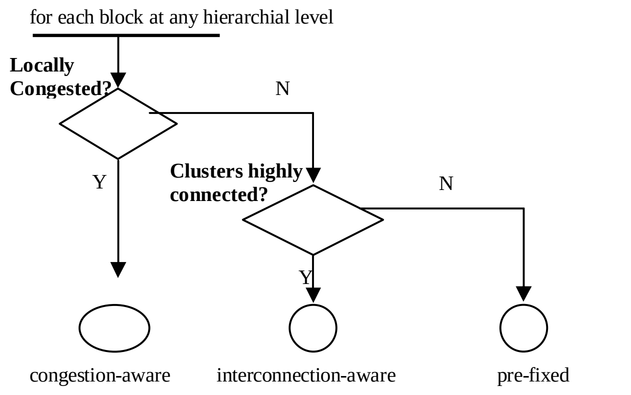
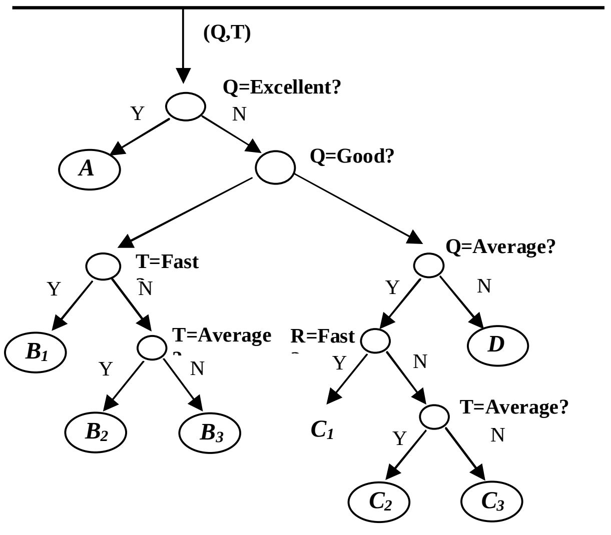


Fig. 4. Blocking Decision Tree (BDT): decide which blocking scheme to use

The overall flow of the trade-off oriented placement tool is controlled by the global decision tree (GDT) as shown in Fig. 5. The decision of which specific sub-algorithm to use internally in the placement system is made based on the input vector. The user input vector has two elements: final placement quality (Q) and runtime (T). The branches coming out of each node represent different groups of algorithms to use. The leaf node ***A***, ***B_1***, ***B_2***, ***B_3***, ***C_1***, ***C_2***, ***C_3***, ***D*** represent each different approaches we used in this tool to achieve the trade-off between quality and runtime.

In our placement tool, there are four possible values for Q (**E**xcellent, **G**ood, **A**verage, **O**k) and three possible values for T (**F**ast, **A**verage, **O**k). GDT will pick which approach to use based on the input vector {Q, T}.To achieve the trade-off between the quality and runtime, the number of blocks at each hierarchical level is another important factor. The fewer blocks each level has, the longer the runtime is and the better the placement quality is.

Starting at the top node, GDT branches left or right depending on the input vector {Q, T}. When Q is set to Excellent, we do not use any blocking schemes to achieve the best placement quality. When Q is set to Good, approach set B is selected. Approach set B makes use of several blocking schemes described in the previous section to optimize both wirelength and congestion. Specifically, the pre-fixed blocking scheme is used to get the first level blocks. Starting from the second hierarchical level, BDT will be used to automatically select which blocking scheme to use. Each individual approach in the set B (B1, B2, B3) differs from each other by the number of blocks it has at each hierarchical level. The more blocks an approach has, the faster the approach is.



***Q={E**xcellent, **G**ood, **A**verage, **O**k}*
T={***F***ast, ***A***verage, ***O***k}

Fig. 5. Global Decision Tree (GDT): A decision tree to control the overall flow of our trade-off oriented placer.

When Q is set to Average, approach set C is selected. Approach set C uses the pre-fixed blocking scheme at both the first and the second hierarchical level. BDT is used after the second hierarchical level. Similar to the approach B set, different approaches in the C set differs themselves by the number of blocks at each hierarchical level. When Q is set to Ok, approach D is selected. Approach D is to use the pre-fixed blocking scheme at all the hierarchical levels and each block only contain one bin. It is essentially the same algorithm as the classical top-down partitioning placement algorithm.

To verify the trade-off between quality and runtime exists in our decision tree, we tested our placer with all input vectors on IBM placement benchmarks. Fig. 6 shows the runtime vs. wirelength curve for each circuit we tested. The x axis is runtime in a unit of 100 seconds; the y axis is the total wirelength. From Fig. 6 we can see that different approaches used in our placement tools can indeed control the trade-off between runtime and quality. The approach which runs longer produces a final placement with the highest quality (smallest total wirelength). .

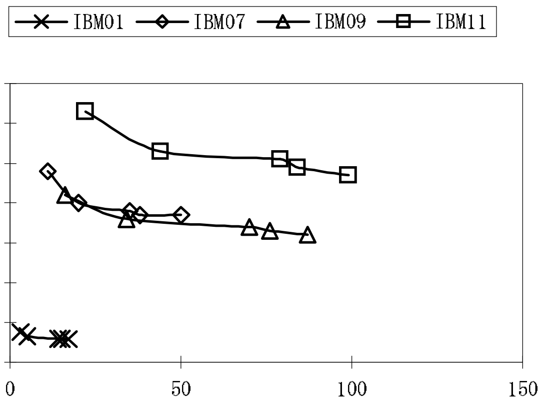


Fig. 6. Placement wirelength vs. runtimes for different benchmarks and input vectors

4 EXPERIMENTAL RESULTS

Table 4 shows the placement results comparison between TOOP and QPlace. Even the fastest mode of TOOP produces a similar or better placement result comparing to QPlace.

Table 4. Improvement of TOOP over QPlace on placement, routing wirelength and number of vias

	TOOP(Q=E)			TOOP (Q=G)			TOOP (Q=O)		
	WL (P)	WL (R)	vias	WL (P)	WL (R)	vias	WL (P)	WL (R)	vias
ibm01	17%	8%	-2%	16%	3%	-4%	10%	4%	-5%
ibm07	4%	11%	11%	-2%	3%	7%	-8%	6%	7%
ibm08	9%	11%	16%	3%	4%	12%	-6%	3%	12%
ibm09	10%	11%	15%	5%	7%	4%	1%	4%	2%
ibm10	8%	10%	14%	1%	4%	1%	-9%	-4%	-3%
ibm11	8%	13%	3%	4%	5%	0	-1%	-1%	0
Aver.	9%	11%	7%	5%	4%	3%	-1%	2%	2%

*For ibm08, the routing result is considered *finished* with 2 violations.

Compared to Capo (Table 5), TOOP shows significantly stronger ability to produce routable placement. Capo fails to produce routable placement (without routing violation) for ibm01, ibm07 and ibm08. For other tested circuits, even our lowest quality mode in TOOP produces placement with better routing wirelength (by 1%) and fewer number of vias (by 4%)

Table 5. Improvement of TOOP over Capo on placement, routing wirelength and number of vias

	TOOP(Q=E)			TOOP (Q=G)			TOOP (Q=O)		
	WL (P)	WL (R)	vias	WL (P)	WL (R)	vias	WL (P)	WL (R)	vias
ibm01	6%	-	-	6%	-	-	0	-	-
ibm07	9%	-	-	5%	-	-	-2%	-	-
ibm08	7%	-	-	1%	-	-	-6%	-	-
ibm09	8%	8%	6%	3%	4%	5%	-1%	1%	3%
ibm10	10%	14%	10%	4%	8%	8%	-6%	1%	4%
Aver.	8%	11%	8%	4%	6%	7%	-3%	1%	4%

.

5 CONCLUSION

In this paper, we developed a trade-off oriented placement tool (TOOP) which can self-adjust based on user’s requests and netlist properties. Decision trees are used internally to control TOOP’s placement process.

Experimental results show that TOOP can indeed get a nice trade-off between different modes. By comparing with Cadence QPlace and an academic placement tool Capo, TOOP shows strong ability in producing high quality and routable placement results. Even the lowest quality mode of TOOP can produce similar or better placement results.

REFERENCES

[1] S. N. Adya, I. L. Markov, P. G. Villarrubia, P. Parakh, M. C. Yildiz, and P. H. Madden, "Benchmarking for Large-Scale Placement and Beyond," *International Symposium on Physical Design*, April 2003.

[2] A. E. Caldwell, A. B. Kahng, and I. L. Markov. “Can Recursive Bisection Alone Produce Routable Placements?”. In *Design Automation Conferenc*e, pages 477–482. IEEE/ACM, June 2000.

[3] C.-C. Chang, J. Cong and M. Xie, "Optimal Scalability Study of Existing Placement Algorithms," Asia South Pacific Design Automation Conference, Kitakyushu, Japan, pp 621-627, January 2003.

[4] S. Hur and J. Lillis. “Mongrel: Hybrid Techniques for Standard Cell Placement”. In *International Conference on Computer-Aided Desig*n, pages 165–170. IEEE, 2000.

[5] M. Wang, X. Yang, and M. Sarrafzadeh. “Dragon2000: Fast Standard-cell Placement for Large Circuits”. In *International Conference on Computer-Aided Desig*n, pages 260–263. IEEE, 2000.

[6] X. Yang, R. Kastner, and M. Sarrafzadeh. “Congestion Reduction During Placement Based on Integer Programming”. In *International Conference on Computer-Aided Desig*n, pages 573–576. IEEE, 2001.

[7] M. C. Yildiz and P. H. Madden. “Improved Cut Sequences for Partitioning Based Placement”. In *Design Automation Conferenc*e, pages 776–779. IEEE/ACM, 2001.