Wirelength Minimization for Min-Cut Placements via Placement Feedback

Andrew B. Kahng, *Member, IEEE*, and Sherief Reda, *Student Member, IEEE*

***Abstract*—The advent of strong multilevel partitioners has made top–down min-cut placers a favored choice for modern placer implementations. Terminal propagation is an important step in min-cut placers because it translates partitioning results into global-placement wirelength assumptions. In this work, the repartitioning problem is carefully reexamined (*Proc. ACM/IEEE Int. Symp. Physical Design*, p. 18, 1997) in the context of terminal propagation and studied in an in-depth manner. Abstractly, it was observed that in repartitioning, future cell locations are used for present terminal propagations and that this can be conceptually regarded as a form of placement feedback. This concept was utilized to achieve accurate terminal propagation via feedback iteration and controller insertion to fine-tune the feedback response. This yields substantial reductions in placement wirelength. Implementing our approach in Capo [version 8.7 (*Proc. ACM/IEEE Design Automation Conf.*, p. 477, 2000 and *GSRC Bookshelf*)] and applying it to standard benchmark circuits yields up to 14% wirelength reductions for the IBM benchmarks with an average improvement of 5.5% and up to 10% reductions for the Peko benchmarks with an average improvement of 5.37%. Experiments also show consistent improvements for routed wirelength, yielding up to 9% wirelength reductions and 5.8% average reduction with acceptable increase in placement runtime. In practice, the method proposed significantly improves routability without building congestion maps and also reduces the number of vias.**

***Index Terms*—Min-cut partitioning, routing, terminal propagation, VLSI placement.**

I. Introduction

RECENT studies on placement optimality [7]–[9], indicate that current placer-solution quality might not be close to optimal. This apparent performance gap needs to be addressed. Recently, top–down min-cut placers have become a favored choice for modern placer implementations [5], [19], [22]. This choice is mainly motivated by the availability of strong multilevel partitioners, as well as the excellent scalability and runtime promise of the top–down paradigm. However, the aforementioned studies demonstrate that a performance and scaling gap exists for top–down min-cut placers—but not for multilevel partitioners, which are the main engines for top–down min-cut placers. This raises the question of how excellent partitioner performance may be "mistranslated" into far-from-excellent placement performance.

To understand this question, one must examine the main components—apart from multilevel partitioners—that determine a min-cut placement result. These components include: 1) top–down paradigm; 2) cut-sequence; and 3) terminal propagation. If we examine the first component, i.e., the top–down paradigm, then an obvious question is whether a 2^k-way partitioner gives far better results than executing a 2-way partitioner for k levels. This has been answered in the negative by Karypis and Kumar [14]. The second component, cut-sequences, have enjoyed much recent attention [2], [5], [21]. Caldwell *et al.* [5] suggest using block aspect ratio as the decisive factor in determining cut direction; this leads to flexible slicing-floorplan structures rather the traditional horizontal–vertical alternation. Block aspect ratio has also been explored via a dynamic-programming-based approach [21] and fractional cut sequences [2], which lead to further reductions in wirelength. The third item, terminal propagation, has not enjoyed much investigation yet is decisive, since it is responsible for translating the partitioner results into global-placement wirelength assumptions. Few works address terminal propagation [5], [6], [11], [13], [17], [19], and mostly follow the initial approach of Dunlop and Kernighan [11]. Other approaches try to omit terminal propagation altogether and opt for global or exact wirelength objectives [13], [22], [23]. Accurate terminal propagation is the subject of this work.

We define ambiguous terminal propagations as propagations arising from terminals that lie equally proximate from two subblocks of a block being partitioned, so that their destination propagation is ambiguous. To reduce this ambiguity, we carefully reexamine the repartitioning problem [13] and show that it is abstractly a form of placement feedback, where future cell locations are used to determine present terminal-propagation results. Since these terminal propagations produce new results that change the output results, the feedback can be iterated a number of times in order to attain stable and consistent improvements. We propose and investigate variant "feedback controllers" to fine-tune the placement response and optimize wirelength. We summarize our contributions as follows.

1) We reexamine the repartitioning problem [13] (without overlapping) in the context of top–down recursive-bisection placement and quantify its effect on the number of ambiguous propagations.
2) We show that the problem is similar to feedback systems.
3) We propose to iterate the number of repartitions according to a number of different objectives, i.e., controllers.
4) We develop efficient implementations.

Manuscript received June 11, 2004; revised December 15, 2004 and April 17, 2005. Earlier results of this work were presented at the Design Automation Conference 2004. This paper was recommended by Associate Editor M. D. F. Wong.

A. B. Kahng is with the Departments of Computer Science and Engineering, and of Electrical and Computer Engineering, University of California (UC) San Diego, La Jolla, CA 92093-0114 USA (e-mail: abk@ucsd.edu).

S. Reda is with the Department of Computer Science and Engineering, University of California (UC) San Diego, La Jolla, CA 92093-0114 USA (e-mail: sreda@cs.ucsd.edu).

Digital Object Identifier 10.1109/TCAD.2005.855917

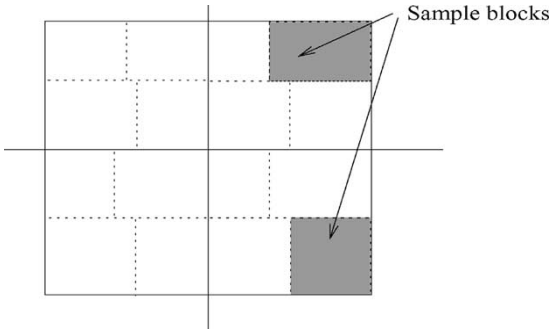

Fig. 1. Snapshot of a min-cut placement. Solid horizontal lines represent first-level cuts, and solid vertical lines represent second-level cuts. Dashed horizontal lines represent third-level cuts, and dashed vertical lines represent fourth-level cuts.

The organization of this paper is as follows. In Section II, we examine the top–down min-cut placement methodology and its essential component of terminal propagation. In Section III, we present our feedback methodology for accurate terminal-propagation control. Section IV gives experimental results on various standard benchmarks. Finally, Section V summarizes our work and presents directions for future work.

II. Background

In this section, we give a brief overview of top–down min-cut placement as well as the necessary background for terminal propagation.

A. Top–Down Min-Cut Placement

In min-cut placement, a placement region is a collection of blocks. Each block corresponds to a fixed rectangle into which nodes of a hypergraph should be placed. Initially, the chip's core region is comprised of one block. The min-cut placement methodology proceeds by recursively partitioning each block and its associated hypergraph, and assigning the partitioned subhypergraphs to subblocks. All nodes (or cells) that are assigned to a subblock are considered, for wirelength-estimation and terminal-propagation purposes, to be placed at the geometric center of the block. Partitioning usually alternates between vertical and horizontal cuts, or as determined by the block aspect ratio [5], [18], [21]. The product of the partitioning process is a slicing floorplan as shown in Fig. 1. The partitioning process continues until a certain block threshold size, beyond which end-case placers [4] are used to assign actual locations of hypergraph nodes in their corresponding blocks. Given a set of disjoint blocks whose union is the entire placement region, we use the term placement-level partitioning to indicate the process of partitioning each block exactly once. Hence, the whole min-cut top–down placement methodology can be considered as the progression of placement levels from a coarse top level down to a fine bottom level.

B. Terminal Propagation

Terminal propagation [11] is the process through which nodes external to a given block under partition are propagated as fixed terminals (nodes) to that block. These terminals bias the partitioner toward placing movable nodes close to their terminals, thus reducing placement wirelength. Given a block under partition to two subblocks and a node externally connected to this block, the subblock to which this node is propagated as a terminal is typically determined by: 1) calculating the distances between the node's position and the centers of the two new subblocks; and 2) with some tolerance, propagating the node to the closer center as a fixed terminal.

Consider some block B being partitioned into two subblocks B_1 and B_2, with the geometrical center of each block denoted by $c(B_1)$ and $c(B_2)$. Given a net L with some set of cells L_i in B and some set of cells L_e in other blocks, we can partition L_e into three sets.

1) $L_e^1 \subseteq L_e$ is the set of cells that are geometrically closer to $c(B_1)$ than $c(B_2)$. We measure the distance in Manhattan norm.
2) $L_e^2 \subseteq L_e$ is the set of cells that are geometrically closer to $c(B_2)$ than $c(B_1)$.
3) $L_e^3 \subseteq L_e$ is the set of cells that are equally proximate to $c(B_2)$ and $c(B_1)$. This is computed with some tolerance, i.e., we consider two distances equal as long as the difference between the two distances does not exceed a certain threshold δ_{fuzzy}.

Given the previous definitions, terminal-propagation decisions can be summarized as follows.

1) Case (1): If $L_e^1 \neq \emptyset$ and $L_e^2 = \emptyset$, then a fixed cell of zero weight is added to the center of B_1 and connected to L_i via a hyperedge.
2) Case (2): If $L_e^1 = \emptyset$ and $L_e^1 \neq \emptyset$, then a fixed cell of zero weight is added to the center of B_2 and connected to L_i via a hyperedge.
3) Case (3): If $L_e^1 \neq \emptyset$ and $L_e^2 \neq \emptyset$, then no terminals are propagated.
4) Case (4): If $L_e^1 = \emptyset$, $L_e^2 = \emptyset$, and $L_e^3 \neq \emptyset$, then no terminals are propagated in this case [1], [11], or one fixed cell is added to B_1, another fixed cell is added to B_2, and both are connected to L_i [5], [6]. We call this case ambiguous terminal propagation.

The following example illustrates terminal-propagation decisions.

Example 1: If a block B is under partition into subblocks B_1 and B_2 as shown in Fig. 2, then any nodes in block C that are connected to nodes in B will be propagated as fixed nodes to B_1 as shown. There is no ambiguity about this propagation, and terminal propagations from any future bisections within block C will continue to be propagated to block B_1. However, for some nodes this cannot be decided accurately. For example, all nodes in block A are equally proximate to both subblock centers of block B. These nodes lead to ambiguous terminal propagation. As indicated earlier, the traditional solution is to propagate such nodes to both subblock centers [5], [6], or not to propagate at all [1], [11]. The intuition behind these propagation approaches is that it is better to make no decision rather than a bad decision.

Ambiguous terminal propagations can lead to partitioning results that do not capture the global objective of wirelength

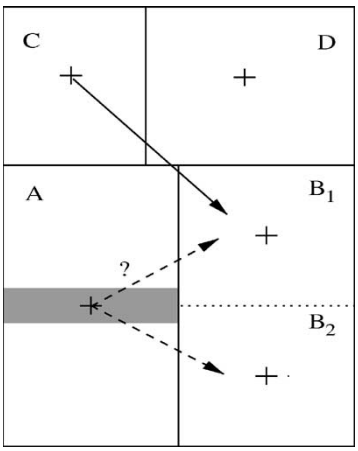

Fig. 2. Example of terminal propagation.

minimization. If $L_e^3 \neq \emptyset$, then the terminal-propagation decision becomes inaccurate: case (1) can likely be case (3), case (2) can likely be case (3), and case (4) can be any of cases (1), (2), or (3). In general, the proximity of a node to a subblock center is calculated with some tolerance δ_{fuzzy} (recently referred to as partition fuzziness in [1]). In Capo [5], this partition fuzziness was originally set to 10%, then later revised to 33% [1]. This latter tolerance matches the value suggested by [11]. The increased fuzziness essentially increases the number of ambiguous terminal propagations to avoid making bad decisions.

To eliminate the dependence of the placement problem on terminal propagation, Huang and Kahng [13] introduced exact objectives (e.g., minimum-spanning-tree models for net routing) to drive the partitioning process. In particular, net vectors are used as means to quantify the global contribution of each cut and to eliminate the need for propagation. Huang and Kahng [13] also introduced the cycling of the partitioning process, by forming a sliding window that goes over the blocks and repartitioning them, since the results of partitioning introduce new terminal locations, and hence, different minimum-spanning-tree costs. The sliding window also overlaps in its movement, allowing cells to migrate from their assigned blocks. Also, Zhong and Dutt [23] and Yildiz and Madden [22] used global half-perimeter wirelength objectives to drive the partitioner. Zhong and Dutt gave experimental results showing improvements versus terminal-propagation-based approaches, at the expense of increased runtime; Yildiz and Madden concluded that wirelength improvements using their approach are modest.

A top–down placement flow using terminal propagation can be conceptually represented as in Fig. 3(a). The input to the placement is the set of nodes initially placed at the center of the core-placement region. Each placement level is divided into two steps: terminal propagation and block partitioning.

III. Accurate Terminal Propagation

A. The Ambiguous Terminal-Propagation Problem

The purpose of this work is to mitigate the effects of ambiguous terminal propagations, i.e., we would like to eliminate or minimize propagations of case (4) when making terminal-propagation decisions. While it may seem that the contribution of these propagations is small in the overall top–down min-cut placement approach, our analyses and results indicate that these propagations can have a tremendous impact on the final wirelength and quality of the min-cut placement. Also, though it might seem possible that reordering block processing can reduce the total amount of ambiguous terminal propagations, our experimental results indicated otherwise. We will later examine the issue of block ordering in Section III-E. We now propose how to mitigate the effects of the ambiguous-terminal-propagation problem using the concept of placement feedback.

B. Placement Feedback

We define placement undoing as merging two subblocks that were originally partitioned, so that they are one block again. Placement undoing enables us to realize accurate terminal propagation. At each level of placement, all blocks are partitioned. After such partitioning, we undo all the partitioned blocks, but we keep the node locations as assigned by the partitioning. That is, we decouple: 1) the placement of a node for use in terminal propagation from 2) its block location. We then use the new accurate node locations to redo block partitioning and update the node locations as necessary, i.e., the output of the placement level is taken back as its input. This can be conceptually regarded as a feedback loop within each placement level, as shown in Fig. 3(b). This feedback takes the current result of a placement level and feeds it back to the input while undoing the placement. Such flow resembles the flow in [2, Fig. 1]. The following example provides an illustration.

Example 2: If block B is under partition into two subblocks B_1 and B_2 as shown in Fig. 2, then we partition block B, propagating nodes in block A to both B_1 and B_2 (ambiguous propagation). We then partition block A (into two subblocks A_1 and A_2), as well as blocks C and D. Now that the whole placement level is partitioned, we undo all block partitionings, restoring the original structure. Despite our having undone the partitioning, we keep the node locations as given by the partitioning results. We use these new locations as input to redo the partitioning, where in this case, no ambiguous terminal propagation occurs. The final node locations are adjusted according to the redone partitioning results.

We stress that feedback only alters the terminal-propagation results of ambiguous propagations. For example, the propagation locations of nodes propagated from block C to B will not change when we apply feedback. It is only ambiguous propagations from block A to B that benefit from such feedback.

We empirically examine the relation between reductions in ambiguous terminal propagations and wirelength reduction as measured by half-perimeter wirelength (HPWL). We implement placement feedback in a well-established top–down min-cut placer, Capo (Version 8.7 [5], [12]). Our changes take 130 lines of code. We report two metrics: 1) the percentage reduction in ambiguous terms per placement level,

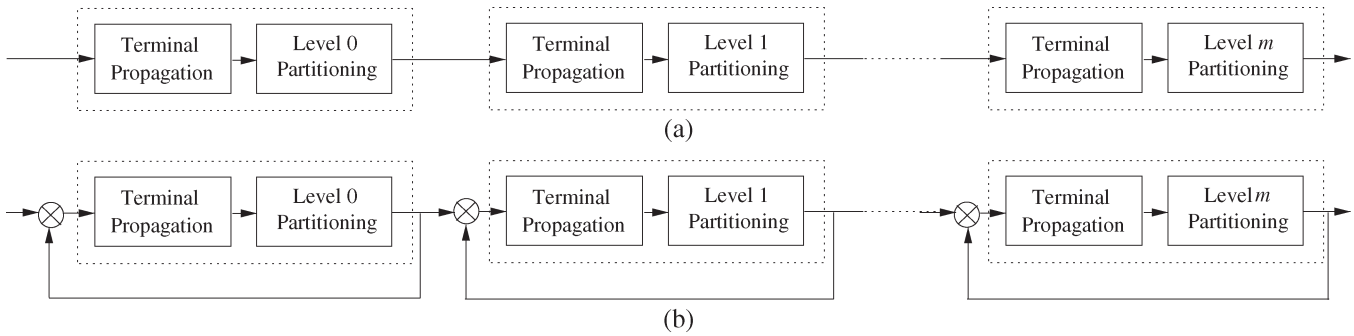

Fig. 3. View of the placement process. (a) Traditional placement flow. (b) Proposed placement flow with feedback loops for accurate control of terminal propagation.

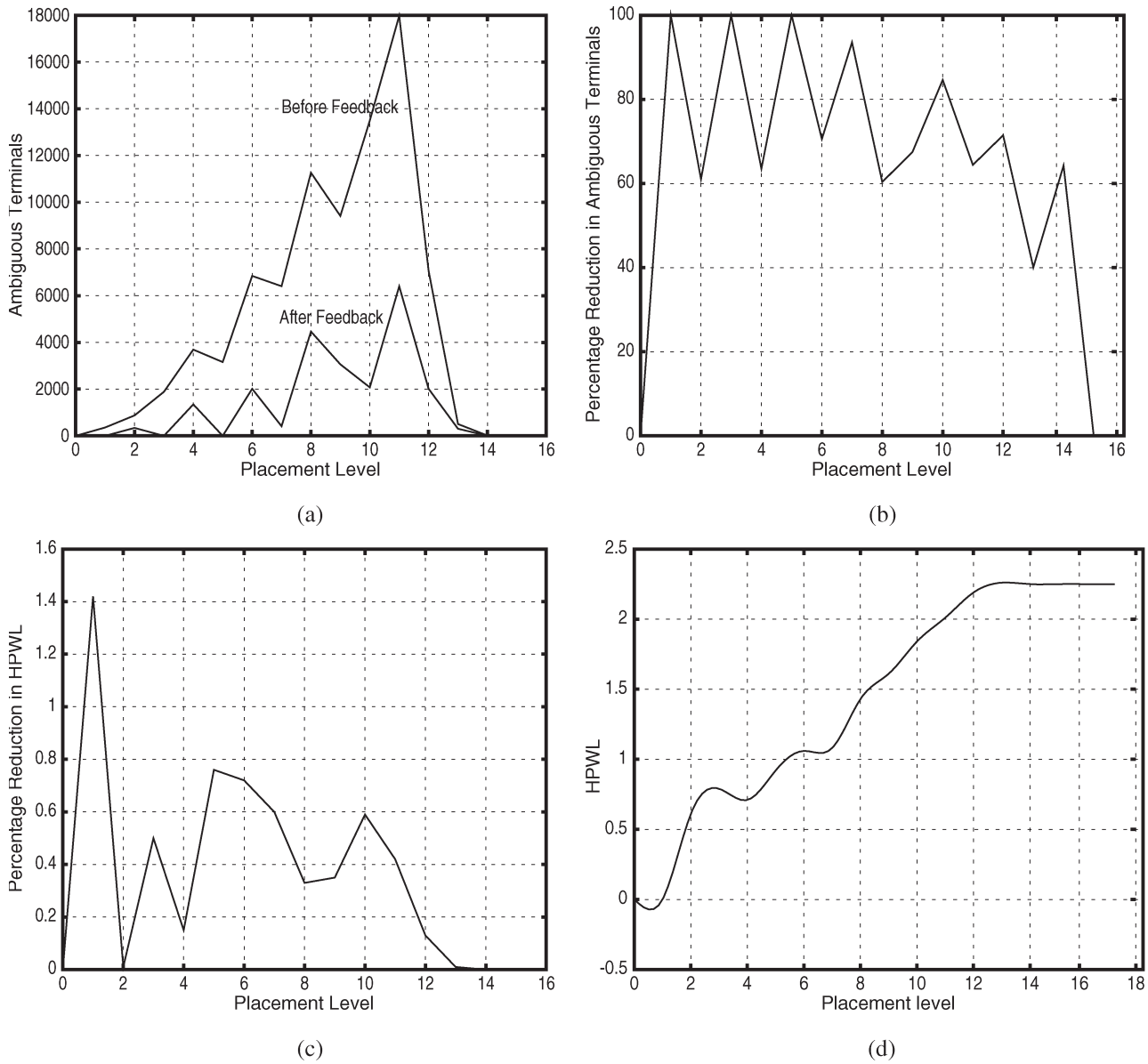

Fig. 4. Relation between wirelength and ambiguous terminal reduction and placement level. (a) Actual values of ambiguous terms before and after feedback for each placement level. (b) Percentage reduction in ambiguous terminals for each placement. (c) Percentage reduction in wirelength for each placement level. (d) Cumulative percentage reduction in wirelength after each placement level.

i.e., we calculate the number of ambiguous terminals before and after feedback; and 2) the percentage reduction of HPWL per placement level, i.e., we calculate the percentage reduction in the HPWL estimate of each level (assuming, as is standard, pin locations at block centers). For the ibm01 benchmark [12], we report the actual number of ambiguous terminal

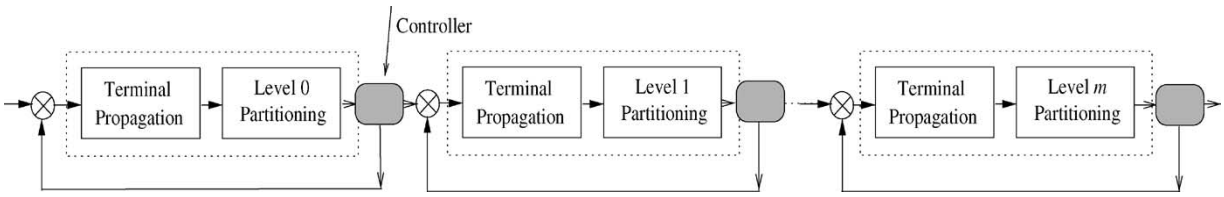

Fig. 5. Feedback system with controllers.

propagations before and after feedback in Fig. 4(a), the percentage reduction in ambiguous terminals in Fig. 4(b), and the percentage reduction in HPWL in Fig. 4(c). The two percentage reductions in Fig. 4(b) and (c) are well correlated with each other, lending support to our intuition. The total number of ambiguous propagations across all placement levels drops from 82 947 terminals to 22 435 terminals, a reduction of around 73%. In another experiment, we quantify the contribution of each level feedback loop to the final HPWL. To do this, given a level i, we enable the feedback loops for all levels up to level i and disable all the remaining $m - i$ loops, where m is the total number of placement levels, and calculate the final HPWL. Our results are given in Fig. 4(d) for all levels of the ibm01 benchmark. These results are averages of six runs with different random seeds. From this figure, we observe that except for the very few last placement levels, reductions in HPWL increase almost linearly with each placement level. We next examine how to fine-tune the placement feedback via the concept of feedback controllers.

C. Iterative Controlled-Placement Feedback

Since the feedback loop produces new outputs, it is natural to iterate over the feedback loop a number of times until one attains the most accurate terminal propagation, and hence, the best overall reduction in HPWL. The problem of feedback systems is that the output might not be predictable, i.e., the system can loop infinitely or, in the best case, converge rapidly to the final stable output [10]. Typically, if the feedback response is not desirable, then some feedback controller is inserted to enhance the response, as shown in Fig. 5. We propose a number of controllers that are suited to the placement problem.

For our purposes, a feedback-controller function controls the response of a placement level by measuring some quality Q at the output of the placement level, and feeding back corrective information to the input of the placement level so as to optimize the quality Q. The controller might also decide to stop sending feedback information, i.e., terminate the looping, if it no longer perceives improvement in the measured quality Q. We propose and motivate two quality measures that can be chosen as objectives during feedback.

1) HPWL quality Q_{H}: Q_{H} is the value of the HPWL estimate at a particular placement level. Since the general placement objective is to minimize wirelength or HPWL, a feedback controller might seek to optimize a placement level based on the HPWL estimate.

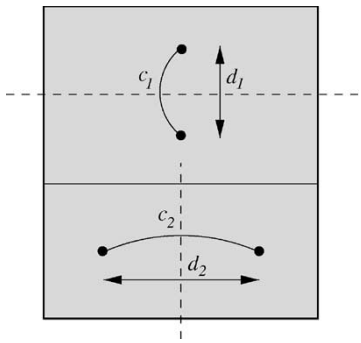

Fig. 6. Discrepancy between the partitioning quality and the HPWL estimate. Partitioning quality is equal to $c_1 + c_2$, while HPWL estimate is equal to $c_1 d_1 + c_2 d_2$.

2) Partitioning quality Q_{P}: Q_{P} is the sum of all partition cuts at a particular placement level. The motivation of this objective is that during the early placement levels, the HPWL estimate (based on block center locations) can be very inaccurate; partitioning results might be more relevant since initial good partitions likely to end up in good final HPWL results.

While it may intuitively seem that optimizing Q_{P} or Q_{H} entails optimizing the other, this might not be the case. The following example illustrates the subtle difference between these two qualities.

Example 3: In Fig. 6, we have two blocks under partition as indicated by the dashed lines. These two blocks are the outcome of an initial partitioning indicated by the horizontal solid line. Assume that after the first partitioning, we obtain cut values c_1 and c_2 as shown in the figure. If the distance between the centers of the child blocks of the top and bottom blocks are d_1 and d_2, respectively, then, $Q_{\mathrm{P}} = c_1 + c_2$ and $Q_{\mathrm{H}} = c_1 d_1 + c_2 d_2$, assuming no connection between the upper and lower blocks. After feedback, these cut values change to c_1' and c_2', and hence $Q_{\mathrm{P}}' = c_1' + c_2'$, while $Q_{\mathrm{H}} = c_1' d_1 + c_2' d_2$. If $c_1' > c_1$, but $c_2' < c_2$, then the change in Q_{H} depends on the values of d_1 and d_2. For example, if $c_1 = 100$, $c_2 = 100$, $d_1 = 6$, and $d_2 = 8$, then $Q_{\mathrm{P}} = 200$ and $Q_{\mathrm{H}} = 1400$, and after feedback $c_1' = 85$, $c_2' = 112$, then $Q_{\mathrm{P}}' = 197$ and $Q_{\mathrm{H}}' = 1406$. Thus, the partitioning quality improves but the HPWL estimate does not improve. A feedback controller that optimizes Q_{H} will definitely choose the first partitioning as the best feedback loop, while a controller that optimizes Q_{P} will choose the second partitioning as the feedback best loop.

In addition to the placement objective to be optimized, a controller must decide when to stop iterating and feed results from its placement level forward to the next placement level.

TABLE I
ITERATIVE-FEEDBACK EXAMPLE. Q_{H} INDICATES THE PLACEMENT-LEVEL QUALITY AS MEASURED BY THE HPWL ESTIMATE. Q_{P} INDICATES THE PLACEMENT-LEVEL QUALITY AS MEASURED BY THE SUM OF PARTITIONING RESULTS

Level	Objective	Iteration					
		0	1	2	3	4	5
2	Q_H	5274440	5227340	5276490	5317450	5278540	5319550
	Q_P	3596	3071	3117	2567	2304	3128
3	Q_H	6382410	8558280	8390840	8341400	8365310	8370650
	Q_P	3991	2277	2188	2160	2189	2166
8	Q_H	13557900	13763900	13727100	13717800	13719900	13727600
	Q_P	6836	6054	5655	5614	5570	5485

We propose and empirically evaluate three kinds of controllers. We assume that each feedback loop is executed for at most k times.

1) Monotonic-improvement controller: In this scheme, the controller fixes some quality to improve, either Q_{P} or Q_{H}, and keeps on iterating over the feedback loop until there is no further improvement in the quality measure, i.e., the controller loops as long as Q_{H} or Q_{P} continues to decrease. The controller stops iterating if an increase in Q_{P} or Q_{H} is observed, and then passes the previous partitioning results to the next placement level.
2) Best improvement controller: In this scheme, the controller fixes some quality to improve, either Q_{P} or Q_{H}, and allows k iterations over the feedback loop. After finishing k loops, the controller passes to the next placement level the results of the best iteration seen (in terms of the chosen quality measure). Notice that the controller does not feed back its best results; it always feeds the current output back to the input. The controller passes the best results seen in k feedback iterations to the next placement level.
3) Unconstrained controller: In this scheme, the controller allows feedback to follow its natural course over the k iterations and then passes the result of the last iteration to the next placement level.

We illustrate the behavior of iterative feedback and the operation of various controllers by the next example.

Example 4: We fix the number of allowable iterations to $k = 5$, and observe a number of placement levels' outputs (as measured by Q_{P} and Q_{H}) for the ibm02 benchmark. We tabulate the results in Table I. The operation of the controllers is illustrated by using the Q_{P} objective. Iteration 0 indicates no feedback loop traversal, i.e., just the regular top–down partitioning. At placement level 2, the monotonic-improvement controller stops at iteration 2, passing the placement result of $Q_{\mathrm{P}} = 3071$; the best improvement controller stops after iteration 5 but passes the best placement result seen ($Q_{\mathrm{P}} = 2304$); and the unconstrained controller passes the last placement of $Q_{\mathrm{P}} = 3128$. At placement level 3, the monotonic-improvement controller stops at iteration 4, passing the placement result of $Q_{\mathrm{P}} = 2160$; the best improvement controller stops after iteration 5 and passes the best placement of $Q_{\mathrm{P}} = 2160$; the unconstrained controller stops after six feedback loops and passes the placement with $Q_{\mathrm{P}} = 2166$.

The last example shows the effect of iterative feedback and controller behavior on individual levels. Further studies examine the final HPWL after all placement levels, i.e., we measure the aggregate effect of all feedback loops and controllers on the final HPWL of the placement. We study the impact of both the allowable feedback iterations and the controller type on the quality of the final placement as measured by HPWL. Results for the same benchmark, ibm02, are plotted in Fig. 7. We plot results of controllers based on the HPWL objective Q_{H} in Fig. 7(a), and results of controllers based on the partitioning objective Q_{P} in Fig. 7(b). Notice that results of the unconstrained controller are identical in Fig. 7(a) and (b). In Fig. 7, the horizontal axis represents the number of feedback iterations; iteration zero represents Capo's results with no feedback. All results represent an average of four seeds. From our experiments, we notice the following.

1) Controllers based on the partitioning objective Q_{P} perform considerably better than controllers based on the HPWL objective Q_{H}. This supports the argument that initial good partitioning results likely lead to final good placements. The HPWL-based controllers are relatively inaccurate at early placement levels and sensitive to the relative distances between blocks. However, they yield better results in the late placement levels. Consequently, we have also tried hybrid approaches by using the Q_{P} objective during the early placement levels and Q_{H} during the late placement levels, but the improvement in quality of results was insignificant.
2) The unconstrained controller performs as well as the Q_{P} best improvement controller since partitioning results tend to improve as the number of feedback iterations increase.
3) The monotonic improvement controller usually takes less runtime since it can iterate for fewer than the k allowed feedback iterations.
4) Overall, after a mere three iterations, the best improvement controller (or unconstrained controller) reduces Capo's final HPWL by about 8%. We have found that the controllers exhibit similar behavior on other benchmarks.

To expand the study of the relationship between final HPWL and number of iterations, we calculate the final HPWL for iterations up to 12, as an approximation to asymptotic analysis. We plot for the ibm02 benchmark the average results of four seeds

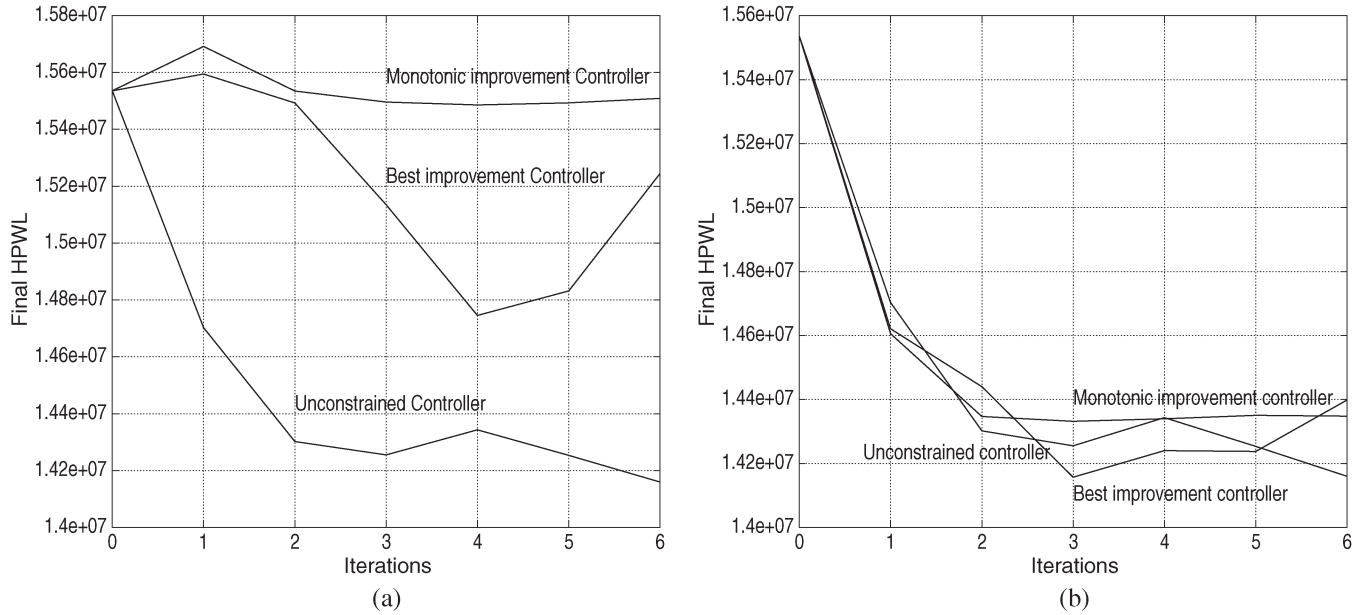

Fig. 7. Effect of controller choice on the final HPWL. (a) Effect of iterative feedback with various controllers using the Q_{H} objective on the final HPWL of the ibm02 benchmark. The horizontal axis represents the number of allowable feedback iterations, while the vertical axis represents the final-placement HPWL. (b) Effect of iterative feedback with various controllers using the Q_{P} objective on the final HPWL of the ibm02 benchmark. The horizontal axis represents the number of allowable feedback iterations, while the vertical axis represents the final-placement HPWL.

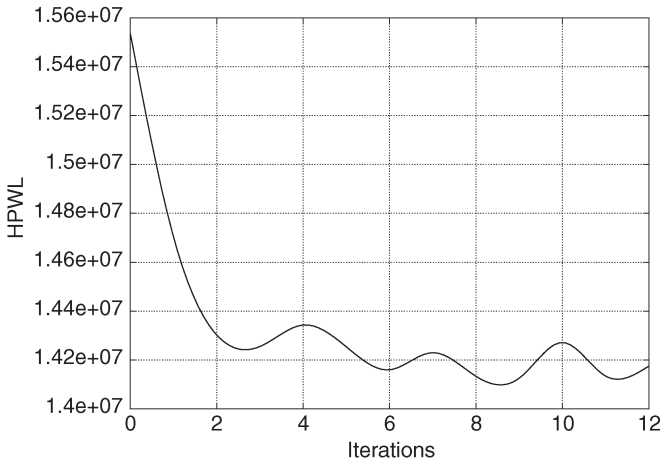

Fig. 8. Effect of iterative feedback using the unconstrained controller on the final HPWL of the ibm02 benchmark. The horizontal axis represents the number of allowable feedback iterations, while the vertical axis represents the final-placement HPWL. The plot shows the average of results of four seeds. Up to 12 iterations are reported as an approximation to asymptotic analysis.

in Fig. 8, smoothing the plot using cubic splines. From the plot, the aggregate final response of the unconstrained controller, as measured by the HPWL estimate, oscillates slightly around a fixed value. We conclude that iterated controlled feedback succeeds in eliminating the indeterminism associated with ambiguous terminal propagations, and transforms the individual placement-level response into an overall stable mechanism that affords 8%–9% HPWL improvement.

D. Accelerated Feedback

A drawback of placement feedback is the increased runtime. We propose a simple idea to reduce the runtime. Typically for each block partitioning, placers execute calls to the multilevel

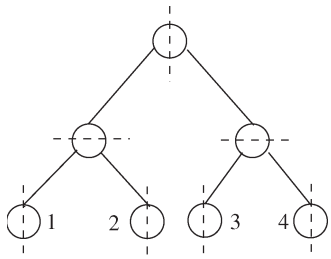

Fig. 9. Block-ordering strategies. Circles represent blocks. Dashed lines represent the cut orientation.

partitioner a number of times and use the best reported results. For instance, Capo calls MLPart [3] twice to construct two cluster trees but only utilizes the best cluster-tree-partitioning results. We notice that in iterated feedback, it is only the last feedback loop that actually determines the partitioning results; other loops determine accurate locations for ambiguous terminals. Hence, in order to speed up our feedback implementation, we call MLPart once for each feedback loop while restoring the default Capo settings for the last feedback iteration. As the experimental results in Section IV demonstrate, this improves runtime considerably.

E. Effect of Block Ordering

We also consider the effect of block ordering on the performance of placement feedback. Our results are inconclusive since they show no improvement in response to a number of various block-ordering strategies. We have tried the following block-ordering strategies.

1) Normal traversal. Process the blocks in the natural order, i.e., as they appear in the top–down tree hierarchy. For example, in Fig. 9, the block ordering would be 1, 2, 3, 4.

TABLE II
RESULTS FOR THE IBM INSTANCES (2% WHITESPACE) FOR SIX RANDOM SEEDS. MODE INDICATES WHETHER RESULTS ARE FOR ORIGINAL CAPO (VERSION 8.7), CAPO WITH ACCELERATED (AFB) FEEDBACK, OR NORMAL (FB) FEEDBACK. FOR ALL INSTANCES, WE USE THE UNCONSTRAINED FEEDBACK CONTROLLER WITH THREE FEEDBACK ITERATIONS. WE REPORT THE BEST AND AVERAGE HPWL RESULTS OF SIX SEEDS. CPU(S) REPRESENTS THE TOTAL CPU TIME IN SECONDS

Cir-cuit	Mode	CPU (s)	Best HPWL	Impr (%)	Average HPWL	Impr (%)	Cir-cuit	Mode	CPU (s)	Best HPWL	Impr (%)	Average HPWL	Impr (%)
ibm01	Capo	102	4.973		5.062		**ibm10**	Capo	740	59.89		61.48	
	+ AFB	130	4.925	0.96	4.990	1.43		+ AFB	1125	58.18	2.85	58.92	4.15
	+ FB	296	4.928	0.92	4.967	1.88		+ FB	2309	58.06	3.05	58.64	4.61
	FengShui	316			4.888			FengShui	2848			57.33	
	Dragon	1615			4.527			Dragon	10839			55.17	
ibm02	Capo	189	15.39		15.53		**ibm11**	Capo	699	45.69		46.44	
	+ AFB	259	14.11	8.31	14.22	8.46		+ AFB	1683	44.46	2.69	44.93	3.25
	+ FB	588	13.94	9.41	14.18	8.67		+ FB	2173	43.30	5.23	43.88	5.51
	FengShui	650			14.06			FengShui	2612			43.82	
	Dragon	2192			13.29			Dragon	8910			40.40	
ibm03	Capo	189	13.74		13.91		**ibm12**	Capo	1011	82.70		83.77	
	+ AFB	425	13.70	0.33	13.80	0.74		+ AFB	2063	75.62	8.56	77.34	7.67
	+ FB	578	13.27	3.47	13.65	1.82		+ FB	3133	76.36	7.67	77.08	7.99
	FengShui	700			12.69			FengShui	3330			75.27	
	Dragon	1938			12.64			Dragon	13926			70.78	
ibm04	Capo	238	17.96		18.17		**ibm13**	Capo	966	54.90		55.96	
	+ AFB	529	16.99	5.41	17.30	4.78		+ AFB	2072	53.64	2.29	54.54	2.54
	+ FB	699	17.07	4.97	17.20	5.35		+ FB	3260	54.19	1.29	54.76	2.15
	FengShui	961			16.43			FengShui	4228			53.43	
	Dragon	2763			16.02			Dragon	11293			50.49	
ibm05	Capo	405	43.52		44.26		**ibm14**	Capo	1642	129.6		131.8	
	+ AFB	648	37.74	13.27	38.31	13.45		+ AFB	4155	122.6	5.44	124.6	5.45
	+ FB	1214	37.63	13.52	38.19	13.73		+ FB	6034	122.1	5.80	122.9	6.79
	FengShui	831			37.51			FengShui	7020			121.3	
	Dragon	5593			36.82			Dragon	21641			117.9	
ibm06	Capo	342	20.75		21.32		**ibm15**	Capo	1708	143.4		145.1	
	+ AFB	636	20.79	-0.17	21.23	0.43		+ AFB	5568	137.2	4.33	138.8	4.39
	+ FB	1135	20.85	-0.49	21.39	-0.32		+ FB	5610	137.2	4.32	138.1	4.82
	FengShui	1042			19.99			FengShui	8235			134.7	
	Dragon	4468			19.19			Dragon	21662			134.4	
ibm07	Capo	483	35.25		35.94		**ibm16**	Capo	2050	185.3		187.0	
	+ AFB	702	32.19	8.68	32.52	9.52		+ AFB	6455	173.3	5.33	175.8	5.65
	+ FB	1493	32.39	8.12	32.64	9.18		+ FB	6735	173.7	6.31	175.1	6.37
	FengShui	1581			30.78			FengShui	12796			179.0	
	Dragon	5030			30.75			Dragon	22657			176.4	
ibm08	Capo	510	38.03		38.21		**ibm17**	Capo	2791	275.5		281.5	
	+ AFB	792	34.51	9.25	34.99	8.43		+ AFB	7207	269.9	0.95	272.7	2.87
	+ FB	1627	34.77	8.57	35.44	7.24		+ FB	8078	267.2	3.01	269.5	4.25
	FengShui	2642			36.79			FengShui	9257			257.9	
	Dragon	7613			32.32			Dragon	25856			248.4	
ibm09	Capo	543	29.69		30.53		**ibm18**	Capo	2657	197.8		199.7	
	+ AFB	730	28.23	4.93	28.74	5.84		+ AFB	4552	191.5	1.30	193.7	1.81
	+ FB	1624	28.39	4.37	28.66	6.11		+ FB	7699	192.0	2.94	192.0	3.82
	FengShui	2857			29.74			FengShui	11611			189.5	
	Dragon	6234			28.26			Dragon	23167			178.4	

TABLE III
RESULTS FOR THE PEKO INSTANCES FOR SIX RANDOM SEEDS. MODE INDICATES WHETHER RESULTS ARE FOR ORIGINAL CAPO (VERSION 8.7), CAPO WITH ACCELERATED (AFB) FEEDBACK, OR NORMAL (FB) FEEDBACK. FOR ALL INSTANCES, WE USE THE UNCONSTRAINED FEEDBACK CONTROLLER WITH THREE FEEDBACK ITERATIONS. WE REPORT THE BEST AND AVERAGE HPWL RESULTS OF SIX SEEDS. CPU(S) REPRESENTS THE TOTAL CPU TIME IN SECONDS

Circuit	Mode	CPU (s)	Best HPWL	Improv (%)	Average HPWL	Impr (%)	Circuit	Mode	CPU (s)	Best HPWL	Impr (%)	Average HPWL	Impr (%)
Peko01	Capo	59	1.39		1.47		**Peko10**	Capo	462	8.47		8.66	
	+ FB	240	1.36	2.23	1.40	5.12		+ FB	1716	7.79	8.02	8.00	7.59
Peko02	Capo	100	2.19		2.24		**Peko11**	Capo	453	8.32		8.89	
	+ FB	404	2.06	6.26	2.11	6.01		+ FB	1754	8.12	2.43	8.61	3.22
Peko03	Capo	118	2.65		2.74		**Peko12**	Capo	587	8.99		9.51	
	+ FB	514	2.49	6.27	2.61	4.88		+ FB	2195	8.41	6.44	8.60	9.60
Peko04	Capo	145	3.08		3.26		**Peko13**	Capo	683	11.04		11.04	
	+ FB	583	2.91	5.47	3.00	7.95		+ FB	2611	9.85	5.12	10.38	5.94
Peko05	Capo	165	3.25		3.33		**Peko14**	Capo	1169	16.49		17.41	
	+ FB	671	3.07	5.36	3.20	4.10		+ FB	4278	15.48	6.10	16.00	8.06
Peko06	Capo	187	3.57		3.72		**Peko15**	Capo	1465	21.17		22.26	
	+ FB	692	3.38	5.47	3.51	5.59		+ FB	5531	20.21	4.50	20.59	7.51
Peko07	Capo	246	5.09		5.30		**Peko16**	Capo	2051	22.77		22.95	
	+ FB	975	4.89	4.26	5.02	5.36		+ FB	7348	22.51	1.15	22.51	1.93
Peko08	Capo	281	5.66		5.92		**Peko17**	Capo	2141	25.07		25.25	
	+ FB	1059	5.57	1.64	5.76	2.81		+ FB	7995	24.57	2.04	24.56	2.72
Peko09	Capo	349	6.39		6.78		**Peko18**	Capo	1434	24.66		25.12	
	+ FB	1280	6.09	4.73	6.42	5.41		+ FB	5063	23.47	4.82	24.29	3.32

2) Random traversal. Process the placement blocks in a random order. One random ordering in Fig. 9 is 1, 4, 3, 2. The intuition behind this approach is to break any adversarial-order dependence to which a fixed ordering might be susceptible.
3) Alternating traversal. Process the ordering normally in one feedback iteration, and then reverse the direction of processing in the next iteration—hence the name, alternate processing. For example, in Fig. 9, one feedback iteration would process the blocks in the order 1, 2, 3, 4, while the next iteration would process the blocks in the reverse order, 4, 3, 2, 1. The intuition behind this approach is to consider all mutual dependencies, i.e., if some block j partitioning results depend on some block i partitioning results, then this dependence will be taken into account in one of the traversal directions.

We have implemented these ordering strategies, but we obtained no better results than those attained by normal traversal.

IV. EXPERIMENTAL RESULTS

Our heuristic is easy to implement and only linearly increases runtime by the number of feedback iterations executed. While there are a number of academic top–down min-cut placers such as Capo [5], Dragon [19], and FengShui [22], we choose to implement our technique in Capo due to its code availability, excellent scalability, fast runtimes, and modular code design. We implement our technique in the latest official release of Capo, version 8.7 (as of November 4, 2003).[1] Our implementation requires 130 lines of C++ code. We report experimental results on four benchmark sets: IBM Version 1 (2% whitespace) [12], Peko [7] (Suite 1), and IBM Version 2 [20] (easy and hard instances). We run our experiments on a 2.4-GHz Xeon Linux workstation with 2-GB memory.

In the first series of experiments, we evaluate our technique on the IBM Version 1 benchmarks [12] (2% whitespace) and give the results in Table II. We report results of original Capo as indicated by Mode Capo, Capo with accelerated placement feedback as indicated by Mode AFB, and normal feedback as indicated by Mode FB. For all experiments, $k = 3$ iterations of unconstrained feedback are used. All runtimes are reported in seconds as indicated by the label CPU (s). We give the best and average of six runs each with a different random seed, and report the percentage improvement in HPWL for both the best and average results.[2] We also report the results of Dragon (version 3.01) [19] and FengShui (version 2.6) [22]. Since there is no direct mechanism to control the random seed of those placers, we report the placement results of only one run. From the table, the average improvements for accelerated feedback and normal feedback are 4.70% and 5.43%, respectively. We

[1]We found a bug that had disabled overlap removal; Capo's authors pointed out how to enable legalization and fix this problem with a trivial amount of coding [16]. We have verified that all of our placement results are legal.

[2]We have enabled rowIroning in both regular and feedback flows. RowIroning is a detailed placer within Capo that optimally replaces windows of seven to eight cells.

TABLE IV
RESULTS FOR THE IBM VERSION 2 INSTANCES (EASY AND HARD). MODE INDICATES WHETHER THE RESULTS REPRESENTS ORIGINAL CAPO'S RESULTS OR CAPO WITH ACCELERATED FEEDBACK (AFB). CPU(S) REPRESENTS THE TOTAL CPU TIME IN SECONDS. FOR ROUTABILITY RESULTS, WE REPORT ROUTING CPU TIME IN SECONDS, NUMBER OF ROUTING VIOLATIONS, VIAS, AND ROUTED WIRELENGTH (WL). IMPR INDICATES THE IMPROVEMENT PERCENTAGE IN WIRELENGTH FOR FEEDBACK OVER CAPO. ALL PLACEMENTS WERE ROUTED USING THE LINUX VERSION OF CADENCE'S WARPROUTE 2.4

Circuit	Mode	Place			Route				
		CPU(s)	HPWL	Impr	CPU(s)	Viol	Vias	WL	Impr
ibm01e	Capo	57	5.590		10800	1238	147426	928179	
	+ AFB	195	5.291	5.97%	6051	0	140353	872629	6.39%
	Dragon	1613	5.739		1447	0	147467	902624	
	QPlace	173	5.930		840	0	143513	944838	
ibm01h	Capo	77	5.514		10830	601	148416	909431	
	+ AFB	203	5.298	3.91%	7964	103	146437	895000	1.54%
	Dragon	1575	5.537		1709	0	134582	859441	
	QPlace	178	5.637		1114	0	144539	890141	
ibm02e	Capo	133	15.76		1411	0	311617	2371683	
	+ AFB	355	15.10	4.18%	951	0	298034	2228700	6.75%
	Dragon	2273	15.95		2160	0	311270	2234878	
	QPlace	390	15.55		638	0	290097	2146602	
ibm02h	Capo	149	15.39		2138	0	308345	2240272	
	+ AFB	292	14.85	3.50%	2365	0	309148	2222011	0.90%
	Dragon	2417	14.72		1482	0	302684	2215226	
	QPlace	391	15.23		1365	0	304747	2211047	
ibm07e	Capo	403	36.64		2734	0	583716	4953781	
	+ AFB	666	34.96	5.33%	1919	0	565161	4617930	6.86%
	Dragon	3899	36.60		2474	0	559697	4541095	
	QPlace	730	37.21		1864	0	562852	4581759	
ibm07h	Capo	459	35.16		11501	450	611954	5124405	
	+ AFB	705	34.95	0.59%	3160	0	575019	4657547	9.17%
	Dragon	3947	34.38		3201	0	582867	4697263	
	QPlace	712	36.97		1850	0	600736	5043070	
ibm08e	Capo	446	38.82		1714	0	682384	5145286	
	+ AFB	966	37.04	4.58%	1215	0	660089	4770640	7.75%
	Dragon	8586	36.10		997	0	662764	4514431	
	QPlace	994	39.46		914	0	703677	5267644	
ibm08h	Capo	452	38.48		7560	59	726584	5213489	
	+ AFB	984	36.43	5.32%	1180	0	669970	4841286	7.10%
	Dragon	8606	34.54		930	0	674494	4466114	
	QPlace	1012	38.50		1414	0	724930	5338340	

also observe that HPWL improvements peak at nearly 14% for ibm05. Comparing runtimes, we find that accelerated feedback increases runtime to 2.02x Capo, and that feedback increases runtime to 3.13x Capo. We conclude that accelerated feedback significantly improves runtime with a small impact on solution quality as measured by HPWL.

In the second series of experiments, we evaluate our technique on the Peko benchmarks [7] and give the results in Table III. The column descriptions are identical to those of Table II. We notice that original Capo results as reported by [7] are different from the ones we report due to the release of a new version of Capo (version Capo 8.7 [1] rather than the old version 8.0 used by [7]). We can see that our technique successfully obtains a further reduction in wirelength by up to 10%, with an average improvement of 5.37%, versus the latest Capo results.

Our third series of experiments evaluates the impact of our heuristic on both routability and final routed wirelength of the IBM version 2 [20] benchmarks by using Cadence's WRoute Version 2.4. We report the experimental results in Tables IV and V for both IBM version 2 easy and hard instances. We also report the results of both Dragon and Cadence's QPlace[3] for the sake of comparison. Placements of other tools, e.g.,

[3]Only QPlace placement runtimes are reported on a Solaris Ultra 10 machine.

TABLE V
RESULTS FOR THE IBM VERSION 2 INSTANCES (EASY AND HARD). MODE INDICATES WHETHER THE RESULTS REPRESENTS ORIGINAL CAPO'S RESULTS OR CAPO WITH ACCELERATED FEEDBACK (AFB). CPU(S) REPRESENTS THE TOTAL CPU TIME IN SECONDS. FOR ROUTABILITY RESULTS, WE REPORT ROUTING CPU TIME IN SECONDS, NUMBER OF ROUTING VIOLATIONS, VIAS, AND ROUTED WIRELENGTH (WL). IMPR INDICATES THE IMPROVEMENT PERCENTAGE IN WIRELENGTH FOR FEEDBACK OVER CAPO. ALL PLACEMENTS WERE ROUTED USING THE LINUX VERSION OF CADENCE'S WARPROUTE 2.4

Circuit	Mode	Place			Route				
		CPU(s)	HPWL	Impr	CPU(s)	Viol	Vias	WL	Impr
ibm09e	Capo	295	3189810		2657	1	532641	3413247	
	+ AFB	1029	3134550	1.88%	3632	0	532894	3468914	-1.61%
	Dragon	9158	3011756		3235	0	560756	3566572	
	QPlace	805	3388074		4535	0	545424	3595004	
ibm09h	Capo	315	3253120		2943	0	554942	3578004	
	+ AFB	922	3092060	4.94%	4520	0	550386	3528053	1.40%
	Dragon	8955	2907006		3057	0	558848	3443702	
	QPlace	890	3501940		5554	0	568012	3861120	
ibm10e	Capo	458	6054520		5556	0	852419	6831436	
	+ AFB	1617	5963430	1.50%	8218	0	849532	6970985	-2.03%
	Dragon	15276	5634591		5931	0	888105	6855377	
	QPlace	1284	6329743		9886	0	879532	7443258	
ibm10h	Capo	483	6317760		15835	0	941380	7771071	
	+ AFB	1636	6156330	2.5%	11242	0	896749	7266408	6.49%
	Dragon	10017	5529089		10407	0	885641	6803801	
	QPlace	1246	6372542		12735	0	921617	7628755	
ibm11e	Capo	469	4699820		3894	0	693248	5063947	
	+ AFB	2053	4664490	0.74%	5968	0	683467	5016880	0.92%
	Dragon	10219	4346040		4672	0	726319	5159810	
	QPlace	1005	5163175		5384	0	733327	6143797	
ibm11h	Capo	659	4693120		3973	0	715129	5213806	
	+ AFB	2775	4581400	2.38%	6165	0	704912	5178964	0.68%
	Dragon	9448	4366608		4256	0	719791	5100862	
	QPlace	1718	4991754		5057	0	745577	5859200	
ibm12e	Capo	488	8668280		33357	43	1107320	10408826	
	+ AFB	1918	8338480	3.81%	17018	0	1055410	10020246	3.65%
	Dragon	18029	7402743		37785	487	1188983	10349721	
	QPlace	1442	9318313		18582	0	1103456	11360578	
ibm12h	Capo	834	8343850		14541	1	1117458	10475534	
	+ AFB	2418	8135740	2.49%	17517	0	1077275	10086777	3.72%
	Dragon	9390	7363033		32793	152	1160365	10314106	
	QPlace	2451	8894402		20797	0	1142057	11109699	

FengShui [22], have been rendered routable by using a postplacement-whitespace-distribution algorithm [15]; we refer the interested reader to [15] for their wirelength results. From the tables, our proposed heuristic improves the routability as measured by the number of violations for all instances. For example, WRoute smoothly routes the feedback placement of the ibm01 easy instance with zero violations. The routability of ibm07 and ibm08 is also dramatically enhanced. These improvements in routability are likely due to the total reductions in wirelength. We also observe that the routing of the feedback placements takes much less time than Capo's placements. Total placement and routing runtime for Capo is 50 864 s, but this drops to 28 901 s with feedback. Hence, the savings in routing time offset any runtime increase in placement due to feedback. For comparison, the total runtime for Dragon is 47 316 s and 14 576 s for QPlace. As for wirelength, we can see that improvements reach up to 9% for ibm07h. The average improvement for routed wirelength of all benchmarks is 5.81% with the best results for the ibm01e and ibm07h testcases. These reductions in wirelength improve total congestion and power consumption. Comparing the number of vias, we find that feedback produces the least number of vias in most cases. The total number of vias for Capo is 3416×10^3, and 3362×10^3 vias with feedback (3371×10^3 vias for Dragon and 3470×10^3 for QPlace). These reductions in the number of vias, albeit slightly, may improve both manufacturing yield and total delay.

V. Conclusion

In this paper, we have studied the problem of ambiguous terminal propagations that introduces indeterminism in the placer performance. We have carefully reexamined the repartitioning problem [13] to diminish this indeterminism. We abstractly identified repartitioning as a form of placement feedback. In feedback, future node locations control present terminal propagation. This is realized by undoing a placement level after its partitioning, and feeding back the resultant node locations to the same placement level as input for partitioning. This feedback scheme is iterated, where a number of variant controllers to fine-tune the feedback response are proposed and compared. Implementing our approach in Capo and applying it to standard benchmarks yields up to 14% HPWL reductions (average 5.5%) for the IBM general benchmarks (Version 1), up to 10% HPWL reductions (average 5.37%) for the Peko (Suite 1) benchmarks, and up to 9% routed wirelength reductions (average 5.8%) for the IBM (Version 2) benchmarks. In addition, due to the reduction in wirelength, the proposed approach improves routability, the routing runtime, and the number of vias. Our future work will focus on further runtime improvements.

References

[1] S. Adya, I. Markov, and P. Villarrubia, "On whitespace and stability in mixed-size placement," in *Proc. IEEE Int. Conf. Computer-Aided Design*, San Jose, CA, 2003, pp. 311–317.

[2] A. Agnihotri, M. Yildiz, A. Khatkhate, A. Mathur, S. Ono, and P. Madden, "Fractional cut: Improved recursive bisection placement," in *Proc. IEEE Int. Conf. Computer-Aided Design*, San Jose, CA, 2003, pp. 307–310.

[3] C. J. Alpert, J. H. Huang, and A. B. Kahng, "Multilevel circuit partitioning," in *Proc. ACM/IEEE Design Automation Conf.*, Anaheim, CA, 1997, pp. 530–533.

[4] A. Caldwell, A. Kahng, and I. Markov, "End-case placers for standard-cell layout," in *Proc. ACM/IEEE Int. Symp. Physical Design*, Monterey, CA, 1999, pp. 90–96.

[5] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Can recursive bisection alone produce routable placements," in *Proc. ACM/IEEE Design Automation Conf.*, Los Angeles, CA, 2000, pp. 477–482.

[6] ——, "Optimal partitioners and end-case placers for standard-cell layout," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 19, no. 11, pp. 1304–1313, Nov. 2000.

[7] C. Chang, J. Cong, and M. Xie, "Optimality and scalability study of existing placement algorithms," in *Proc. IEEE Asia and South Pacific Design Automation Conf.*, Kitakyushu, Japan, 2003, pp. 621–627.

[8] J. Cong, M. Romesis, and M. Xie, "Optimality and scalability study of partitioning and placement algorithms," in *Proc. ACM/IEEE Int. Symp. Physical Design*, Monterey, CA, 2003, pp. 88–94.

[9] J. Cong, T. Kong, J. R. Shinnerl, M. Xie, and X. Yuan, "Large-scale circuit placement: Gap and promise," in *Proc. IEEE Int. Conf. Computer-Aided Design*, San Jose, CA, 2003, pp. 883–890.

[10] R. Dorf and R. Bishop, *Modern Control Systems*, 9th ed. Upper Saddle River, NJ: Prentice-Hall, 2000.

[11] A. E. Dunlop and B. W. Kernighan, "A procedure for placement of standard-cell VLSI circuits," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. CAD-4, no. 1, pp. 92–98, Jan. 1985.

[12] *GSRC Bookshelf*. [Online]. Available: http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/Capo

[13] D. J.-H. Huang and A. B. Kahng, "Partitioning-based standard-cell global placement with an exact objective," in *Proc. ACM/IEEE Int. Symp. Physical Design*, Napa, CA, 1997, pp. 18–25.

[14] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partitioning," in *Proc. ACM/IEEE Design Automation Conf.*, New Orleans, LA, 1999, pp. 343–348.

[15] C. Li *et al.*, "Routability-driven placement and white space allocation," in *Proc. IEEE Int. Conf. Computer-Aided Design*, San Jose, CA, 2004, pp. 394–401.

[16] I. Markov, private communication, Nov. 2003.

[17] P. R. Suaris and G. Kedem, "A quadrisection-based combined place and route scheme for standard cells," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 8, no. 3, pp. 234–244, Mar. 1989.

[18] K. Takahashi, K. Nakajima, M. Terai, and K. Sato, "Min-cut placement with global objective functions for large scale sea-of-gates arrays," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 14, no. 4, pp. 434–446, Apr. 1995.

[19] M. Wang, X. Yang, and M. Sarrafzadeh, "DRAGON2000: Standard-cell placement tool for large industry circuits," in *Proc. IEEE Int. Conf. Computer-Aided Design*, San Jose, CA, 2001, pp. 260–263.

[20] X. Yang, B. Choi, and M. Sarrafzadeh, "Routabtility driven white space allocation for fixed-die standard-cell placement," in *Proc. ACM/IEEE Int. Symp. Physical Design*, San Diego, CA, 2002, pp. 42–47.

[21] M. Yildiz and P. Madden, "Improved cut sequences for partitioning based placement," in *Proc. ACM/IEEE Design Automation Conf.*, Las Vegas, NV, 2001, pp. 776–779.

[22] ——, "Global objectives for standard-cell placement," in *Proc. IEEE Great Lakes Symp. VLSI*, West Lafayette, IN, 2001, pp. 68–72.

[23] K. Zhong and S. Dutt, "Effective partition-driven placement with simultaneous level processing and global net views," in *Proc. IEEE Int. Conf. Computer-Aided Design*, San Jose, CA, 2000, pp. 171–176.

Andrew B. Kahng (A'89–M'03) received the A.B. degree in applied mathematics from Harvard College, Cambridge, MA, and the M.S. and Ph.D. degrees in computer science from University of California (UC) San Diego, La Jolla.

From 1989 to 2000, he was a member of the UC Los Angeles (UCLA) Computer Science Faculty. Since 1997, he has defined the physical-design roadmap for the International Technology Roadmap for Semiconductors (ITRS), and since 2001, has chaired the US and international working groups for Design Technology for the ITRS. He has been active in the Microelectronics Advanced Research Corporation (MARCO) Gigascale Silicon Research Center since its inception. He is a Professor of Computer Science and Engineering (CSE) and of Electronics and Communication Engineering (ECE) at UC San Diego. His research is mainly on the physical design and performance analysis of very-large-scale integration (VLSI), as well as the VLSI design–manufacturing interface. Other research interests include combinatorial and graph algorithms, and large-scale heuristic global optimization. He has published over 200 papers in VLSI computer-aided-design (CAD) literature.

Dr. Kahng has received three Best Paper awards and a National Science Foundation (NSF) Young Investigator Award. He was also the founding General Chair of the Association of Computing Machinery (ACM)/IEEE International Symposium on Physical Design, and cofounded the ACM Workshop on System-Level Interconnect Planning.

Sherief Reda (S'01) received the B.Sc. and M.Sc. degrees in electrical and computer engineering from Ain Shams University, Cairo, Egypt, in 1998 and 2000, respectively. He is working toward the Ph.D. degree at the University of California (UC) San Diego, La Jolla.

He has over 20 refereed publications in the areas of physical design, very-large-scale-integration (VLSI) test and diagnosis, combinatorial optimization, and computer-aided design (CAD) for deoxyribonucleic acid (DNA) chips.

Mr. Reda received a Best Paper Award at Design, Automation and Test in Europe (DATE) 2002 and the First Place Award at the International Symposium on Physical Design (ISPD) 2005 placement contest.