Placement Feedback: A Concept and Method for Better Min-Cut Placements

Andrew B. Kahng
CSE and ECE Departments
University of CA, San Diego
La Jolla, CA, 92093
abk@cs.ucsd.edu

Sherief Reda
CSE Department
University of CA, San Diego
La Jolla, CA, 92093
sreda@cs.ucsd.edu

ABSTRACT

The advent of strong multi-level partitioners has made top-down min-cut placers a favored choice for modern placer implementations. We examine terminal propagation, an important step in min-cut placers, because it is responsible for translating partitioning results into global placement wirelength assumptions. In this work, we identify a previously overlooked problem - *ambiguous terminal propagation* - and propose a solution based on the concept of feedback from automatic control systems. Implementing our approach in Capo (version 8.7 [5, 10]) and applying it to standard benchmark circuits yields up to 14% wirelength reductions for the IBM benchmarks and 10% reductions for PEKO instances. Experiments also show consistent improvements for routed wirelength, yielding up to 9% wirelength reductions with practical increase in placement runtime. In addition, our method significantly improves routability without building congestion maps, and reduces the number of vias.

Categories and Subject Descriptors: B.7.2 [Design Aids]: Placement and routing
General Terms: Algorithms
Keywords: min-cut placement, terminal propagation, feedback

1. INTRODUCTION

Recently, top-down min-cut placers have become a favored choice for modern placer implementations [5, 14, 17]. This choice is mainly motivated by the availability of strong multi-level partitioners, as well as the excellent scalability and runtime promise of the top-down paradigm. Apart from multi-level partitioners, the main components that determine a min-cut placement result include (i) top-down paradigm, (ii) cut-sequence, and (iii) terminal propagation. The focus of this work is the third component, terminal propagation, which is responsible for translating the partitioner results into global placement wirelength assumptions. Few works address terminal propagation [9, 13, 11, 5, 6, 14], and mostly follow the initial approach of Dunlop and Kernighan [9]. Other approaches try to omit terminal propagation altogether and opt for global or exact wirelength objectives [11, 18, 17]. 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 solve this ambiguity, we propose a solution based on the feedback concept from feedback control systems. We use future cell locations to determine present terminal propagation results. Since these terminal propagations produce new results that change the future output result, we iterate the feedback 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:

- We re-examine the repartitioning problem in the context of top-down placement and quantify its effect on the number of ambiguous propagations.
- We show that the problem is similar to feedback systems.
- We propose to iterate the number of repartitions according to a number of different objectives, i.e., controllers.
- We develop efficient implementations.

The organization of this paper is as follows. In Section 2 we examine the top-down min-cut placement methodology and its essential component of terminal propagation. In Section 3 we present our feedback methodology for accurate terminal propagation control. Section 4 gives experimental results on various standard benchmarks. Finally, Section 5 summarizes our work and presents directions for future work.

2. BACKGROUND

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

2.1 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

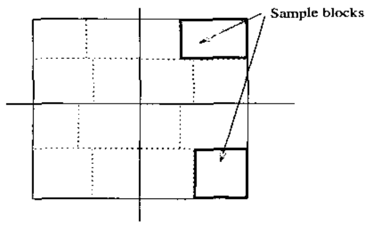


Figure 1: A snapshot of a min-cut placement. Solid horizontal lines represent first level cuts, and solid vertical lines represent second level cuts. Dashed horizontal lines represent third level cuts, and dashed vertical lines represent fourth level cuts.

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, 16]. The product of the partitioning process is a slicing floorplan as shown in Figure 1. The partitioning process continues until a certain block threshold size is reached, beyond which end-case placers [4] are used to assign actual locations of hypergraph nodes within 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.

2.2 Terminal Propagation

Given a block being partitioned, *terminal propagation* [9] is the process through which nodes external to the block being partitioned are propagated as fixed terminals (nodes) to it. These terminals bias the partitioner toward placing movable nodes close to their terminals, hence reducing placement wirelength. Given a block being partitioned 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. The following example illustrates terminal propagation.

Example 1: If a block B is being partitioned into subblocks B_1 and B_2 as shown in Figure 2, then any nodes in block C that are connected to nodes in B will be propagated as fixed nodes to B_1. 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.* The traditional solution is to propagate such nodes to both subblock centers [6, 5], or not to propagate at all [9, 1]. The likely intuition behind these propagation approaches is that it is better to make no decision rather than a bad decision.

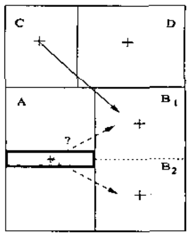


Figure 2: Example of terminal propagation.

We notice that ambiguous terminal propagations lead to indeterminism of the final outcome of a placement level since this depends on the block partitioning order. As mentioned earlier, the proximity of a node to a subblock center is calculated with some tolerance recently referred to as *partition fuzziness* [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 [9]. The increased fuzziness helps to decrease the chance of ambiguous terminal propagations from making bad decisions.

To eliminate the dependency of the placement problem on terminal propagation, Huang and Kahng [11] introduced exact objectives (e.g., minimum spanning tree) 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. [11] also introduces *cycling* of the partitioning process, which entails going over the blocks and repartitioning them since the results of partitioning introduce new terminal locations and hence different minimum spanning tree costs. Also, Zhong and Dutt [18] and Yildiz and Madden [17] used global half-perimeter wirelength objectives to drive the partitioner. [18] gives experimental results showing improvements versus propagation-based approaches at the expense of increased runtime; [17] concludes that wirelength improvements using their approach are modest.

A top-down placement flow using terminal propagation can be conceptually summarized as in Figure 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.

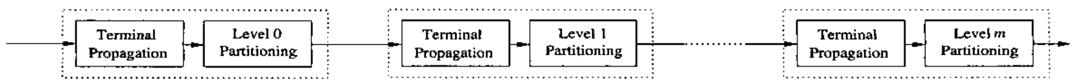
3. ACCURATE TERMINAL PROPAGATION

3.1 The Ambiguous Terminal Propagation Problem

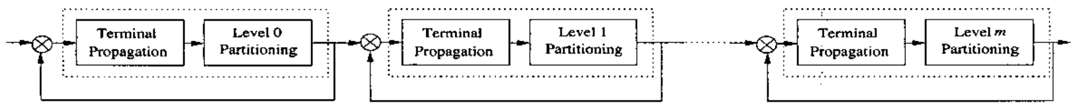
We define the *ambiguous terminal propagation* problem as follows.

Ambiguous Terminal Propagation Problem: Given a current placement level, bisect all blocks with the most accurate possible terminal propagation, i.e., minimize the number of ambiguously propagated terminals.

While one may revert to, e.g., block ordering techniques in an attempt to minimize such ambiguity, our experimental investigation indicates that re-ordering block partitioning does not yield the desired impact. We now examine how to solve the ambiguous terminal propagation problem using the concept of placement feedback.

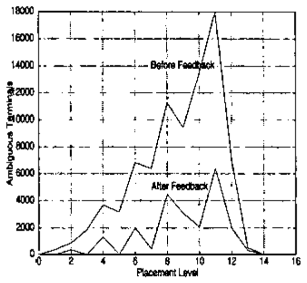


(a) Traditional placement flow

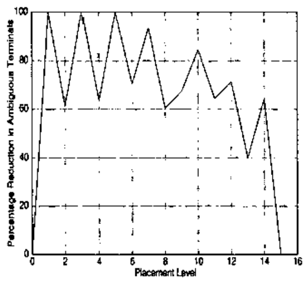


(b) Proposed placement flow with feedback loops for accurate control of terminal propagation

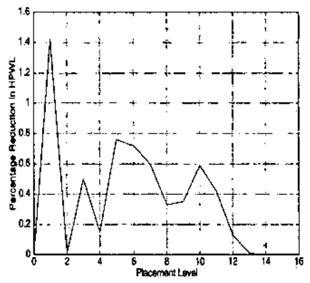
Figure 3: A view of the placement process.



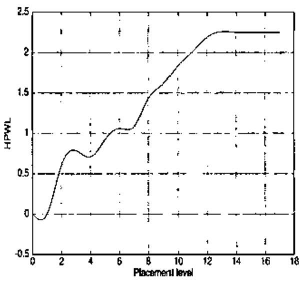
(a) Actual values of ambiguous terms before and after feedback for each placement level.



(b) Percentage reduction in ambiguous terminals for each placement level.



(c) Percentage reduction in wirelength for each placement level.



(d) Cumulative percentage reduction in wirelength after each placement level.

Figure 4: Relation between wirelength and ambiguous terminal reduction and placement level.

3.2 Placement Feedback

In this work, we define *placement undoing* as merging two subblocks that were originally partitioned to be one block. Using placement undoing we can 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 uncouple the placement of a node for use in terminal propagation from its block location. We then use the new accurate node locations to re-do 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 Figure 3(b): this feedback takes the current result of a placement level and feeds it back to the input while undoing the placement. The following example provides an illustration.

Example 2: If block B is being partitioned into two subblocks B_1 and B_2 as shown in Figure 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 re-do the partitioning, where in this case no ambiguous terminal propagation occurs. The final node locations are adjusted according to the re-done partitioning results.

We stress that feedback only alters the terminal propagation results of ambiguous propagations[1]. We now 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, 10]). Our changes take 130 lines of code. We report two metrics: (i) the percentage reduction in ambiguous terms per placement level, i.e., we calculate the number of ambiguous terminals before and after feedback, and (ii) 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 [10], we report the actual number of ambiguous terminal propagations before and after feedback in Figure 4(a), the percentage reduction in ambiguous terminals in Figure 4(b), and the percentage reduction in HPWL in Figure 4(c). We can clearly see that two percentage reductions in Figures 4(b) and 4(c) well-correlate with each other supporting our intuition. The total number of ambiguous propagations across all placement levels drops from 82947 terminals to 22435 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 remaining $m - i$ loops, where m is the total number of placement levels, and calculate the final HPWL. Our results are given in Figure 4(d) for all levels of the ibm01 benchmark. These results are aver-

[1]For example, the propagation locations of nodes propagated from block C to B will not change by using feedback. It is only ambiguous propagations from block A to B that benefit from such feedback, and we use feedback to eliminate the indeterminism associated with ambiguous terminal propagations.

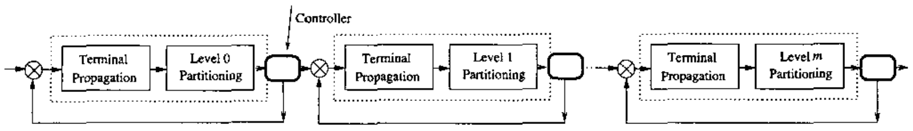


Figure 5: A feedback system with controllers.

Level	Iteration					
	0	1	2	3	4	5
2	0.188	0.220	0.220	0.226	0.229	0.209
3	0.533	0.536	0.531	0.530	0.535	0.526
10	1.373	1.369	1.366	1.365	1.365	1.364

Table 1: Iterative feedback example.

ages of 6 runs with different random seeds. 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.*

3.3 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 with feedback systems is that the output might not be predictable, i.e., the system can loop indefinitely or in the best case converge rapidly to the final stable output [8]. Typically, if the feedback response is not desirable then some *feedback controller* is inserted to enhance the response as shown in Figure 5. We now examine how variant "controllers" can be used to attain the best overall reduction in HPWL.

If we assume that we loop each feedback for at most k times, then to control the response from iterating over the feedback loops, we propose and investigate three variant controllers.

1. *Monotonic Improvement Controller:* In this scheme, the controller keeps on iterating over the feedback loop until there is no further improvement, i.e., as long as the placement's HPWL estimate continues to decrease. The controller stops iterating if an increase in HPWL is observed, and then passes the *previous* partitioning results to the next placement level.
2. *Best Improvement Controller:* In this scheme, the controller allows k iterations over the feedback loop, then passes to the next placement level the results of the *best* iteration seen (in terms of HPWL). Notice that the controller does not feedback its best results; it always feeds the current output back to the input. Rather, it passes the best results seen in k 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 *last* result to the next placement level. From the perspective of an individual level's HPWL, this approach may not give the best HPWL estimate result.

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

Example 3: We set the number of allowable iterations to $k = 5$, and observe a number of placement levels' output (as measured by HPWL estimate) for the ibm02 benchmark. We tabulate the results in Table 1. Iteration 0 indicates no feedback loop traversal. For placement level 2: the monotonic improvement controller stops after 1 feedback loop, passing the placement result of 0.188 HPWL; the best improvement controller stops after 6 feedback loops but passes the placement of the best result seen which is 0.188; and the unconstrained controller passes the last placement with HPWL of 0.209. For placement level 3: the monotonic improvement controller stops after 1 feedback loop passing the placement result of 0.533 HPWL; the best improvement (unconstrained) controller stops after 6 feedback loops and passes the best (last) placement of 0.526. At level 10, the three controllers exhibit the same behavior and report the exact results of HPWL=1.364 of the last feedback iteration.

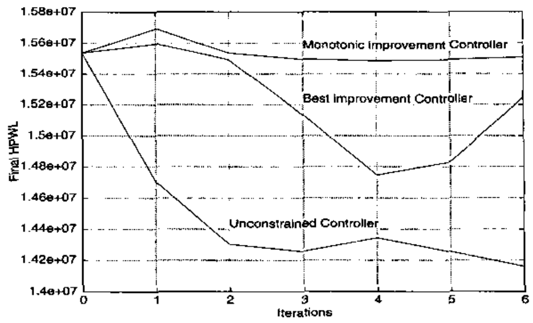


Figure 6: Effect of iterative feedback with various controllers on the *final* HPWL of the ibm02 benchmark. The horizontal axis represents the number of allowable feedback iterations, while the vertical axis represents the final placement HPWL.

The last example shows the effect of iterative feedback and controller behavior on individual levels. The relationship between individual placement level HPWL estimate and the number of feedback iterations does not appear to follow any obvious pattern. We now examine the final HPWL after all placement levels, i.e., we study 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.

We plot our results for the same benchmark, ibm02, in Figure 6. In this figure, the horizontal axis represents the number of feedback iterations, where iteration 0 represents Capo's results with no feedback. All our results are an average of 4 seeds. From the results, we notice that the unconstrained controller produces the best final results. The monotonic improvement controller seems to be poorly performing; on the other hand, the best improvement controller gives HPWL improvements over Capo but is not as good as unconstrained feedback. After a mere 3 iterations, the unconstrained controller reduces Capo's final HPWL by about 8%. We have found that the controllers exhibit similar behaviors on other benchmarks. We believe that the good performance of the unconstrained controller is attributed to the passing of its results of the last feedback iteration to the next placement level. While these results might not be the best in terms of one level's HPWL estimate, they represent an overall more accurate terminal location which translates to global consistent reduction of HPWL. We conclude that the unconstrained controller succeeds in eliminating the indeterminism associated with ambiguous terminal propagations by transforming the individual placement-level response into an aggregate overall stable performance yielding 8-9% HPWL improvement.

3.4 Accelerated Feedback

Typically for each block partitioning, placers execute calls to the multi-level partitioning a number of times and use the best reported results. For instance, Capo calls MLPart [3] twice to construct two cluster trees and only utilize 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 to default Capo settings for the last feedback iteration. As the experimental results demonstrate in Section 4, this improves runtime considerably.

4. 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 [14], and Feng-Shui [17], we decide to implement our technique in Capo due its code availability, excellent scalability, fast runtimes, and modular code design. We implement our technique in Capo, version 8.7[2]. Our implementation efforts require 130 lines of C++ code[3]. We report experimental results on four benchmark sets: IBM Version 1 (2% whitespace) [10], PEKO [7] (Suite 1), and IBM Version 2 [15] (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 [10] (2% whitespace) and give the results in Table 2. 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 6 runs each with a different random seed, and report the percentage improvement in HPWL for both the best and average results. From the table, the average improvements for accelerated feedback and normal feedback are 4.70% and 5.43% respectively. We also observe that HPWL improvements peak at nearly 14% for ibm05. Comparing runtimes, we find that accelerated feedback increases runtime to 2.02× Capo and feedback increases runtime to 3.13× Capo. We conclude that accelerated feedback significantly improves runtime with a small impact on solution quality as measured by HPWL.

In a second series of experiments, we evaluate our technique on the PEKO benchmarks [7]. For space limitations, we omit the detailed results. Results similar to the IBM benchmarks are obtained with an average HPWL improvement of about 5%, and the PEKO12 benchmark attaining up to 10% HPWL improvement.

Our third series of experiments evaluate the impact of our heuristic on both routability and final routed wirelength of the IBM version 2 [15] benchmarks by using Cadence's WRoute Version 2.4. We report the experimental results in Table 3 for both IBM version 2 easy and hard instances. We also report our run results of both Dragon and Cadence's QPlace[4] for sake of comparison; Capo and Dragon are the only two academic placers for which routing results have been reported. Due to the unavailability of Dragon's source code, we are unable to incorporate our techniques into it to estimate the effect of our heuristic on its performance. Other placers like Feng Shui 2.0 [2] produce packed placements by excluding whitespace distribution. The likely outcome of these packed placements is reduced wirelength at the expense of routability [15].

From Table 3, 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 0 violations. Routability of ibm07 and ibm08 is also dramatically enhanced. We stress that routing of the feedback placements takes much less time than Capo's placements. Hence the total placement and routing runtime for Capo is larger than that with feedback. We conclude that the savings in routing time offset any runtime increase in placement due to feedback. As for wirelength, we can see that improvements reach 9% for ibm07. 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 number of vias may improve both the manufacturing yield and total delay.

5. CONCLUSIONS

In this paper we study the problem of ambiguous terminal propagations which introduces indeterminism in the placer performance. We diminish this indeterminism using the concept of *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 placement level as input for partitioning. This feedback scheme is iterated, as is typically done in feedback control systems. We also propose and compare a number of variant controllers to fine-tune the feedback response. Implementing our approach in Capo and applying it to standard benchmark yields up to 14% HPWL reductions for the IBM general instances, 10% HPWL reductions for the PEKO (Suite 1) instances, and 9% actual

[2]We found a bug that had disabled overlap removal; Capo's authors pointed out how to fix this with a trivial amount of coding [12].

[3]Our code modifications are incorporated in the March 2004 release of Capo.

[4]Only QPlace placement runtimes are reported on a Sun Ultra 10 running Solaris 8.

Mode	Cir-cuit	CPU (s)	HPWL	Impr (%)	Cir-cuit	CPU (s)	HPWL	Impr (%)
Capo	**ibm01**	102	5.062		**ibm10**	740	61.48	
+ AFB		130	4.990	1.43		1125	58.92	4.15
+ FB		296	4.967	1.88		2309	58.64	4.61
Capo	**ibm02**	189	15.53		**ibm11**	699	46.44	
+ AFB		259	14.22	8.46		1683	44.93	3.25
+ FB		588	14.18	8.67		2173	43.88	5.51
Capo	**ibm03**	189	13.91		**ibm12**	1011	83.77	
+ AFB		425	13.80	0.74		2063	77.34	7.67
+ FB		578	13.65	1.82		3133	77.08	7.99
Capo	**ibm04**	238	18.17		**ibm13**	966	55.96	
+ AFB		529	17.30	4.78		2072	54.54	2.54
+ FB		699	17.20	5.35		3260	54.76	2.15
Capo	**ibm05**	405	44.26		**ibm14**	1642	131.8	
+ AFB		648	38.31	13.45		4155	124.6	5.45
+ FB		1214	38.19	13.73		6034	122.9	6.79
Capo	**ibm06**	342	21.32		**ibm15**	1708	145.1	
+ AFB		636	21.23	0.43		5568	138.8	4.39
+ FB		1135	21.39	-0.32		5610	138.1	4.82
Capo	**ibm07**	483	35.94		**ibm16**	2050	187.0	
+ AFB		702	32.52	9.52		6455	175.8	5.65
+ FB		1493	32.64	9.18		6735	175.1	6.37
Capo	**ibm08**	510	38.21		**ibm17**	2791	281.5	
+ AFB		792	34.99	8.43		7207	272.7	2.87
+ FB		1627	35.44	7.24		8078	269.5	4.25
Capo	**ibm09**	543	30.53		**ibm18**		199.7	
+ AFB		730	28.74	5.84		4552	193.7	1.81
+ FB		1624	28.66	6.11		7699	192.0	3.82

Table 2: Results for the IBM instances (2% whitespace) for 6 random seeds. Mode indicates whether results are for original Capo (version 8.7), Capo with accelerated feedback (AFB), or normal feedback (FB). For all instances, we use the unconstrained feedback controller with 3 feedback iterations. We report the average HPWL results of 6 seeds. CPU(s) represents the total CPU time in seconds.

Circuit	Mode	CPU	Routing Results			
			Viol	Vias	WL	Impr
ibm01e	Capo	10857	1238	147426	928179	
	+ AFB	6246	0	**140353**	**872629**	6.39
	Dragon	3060	0	147467	902624	
	QPlace	1013	0	143513	944838	
ibm01h	Capo	10907	601	148416	909431	
	+ AFB	8167	103	146437	895000	1.54
	Dragon	3284	0	**134582**	**859441**	
	QPlace	1292	0	144539	890141	
ibm02e	Capo	1544	0	311617	2371683	
	+ AFB	1306	0	**298034**	2228700	6.75
	Dragon	4433	0	311270	2234878	
	QPlace	1028	0	290097	**2146602**	
ibm02h	Capo	2287	0	308345	2240272	
	+ AFB	2657	0	309148	2222011	0.90
	Dragon	3899	0	**302684**	2215226	
	QPlace	1756	0	304747	**2211047**	
ibm07e	Capo	3137	0	583716	4953781	
	+ AFB	2585	0	565161	4617930	6.86
	Dragon	6373	0	**559697**	**4541095**	
	QPlace	2594	0	562852	4581759	
ibm07h	Capo	11960	450	611954	5124405	
	+ AFB	3865	0	**575019**	**4657547**	9.17
	Dragon	7148	0	582867	4697263	
	QPlace	2562	0	600736	5043070	
ibm08e	Capo	2160	0	682384	5145286	
	+ AFB	2181	0	**660089**	4770640	7.75
	Dragon	9583	0	662764	**4514431**	
	QPlace	1908	0	703677	5267644	
ibm08h	Capo	8012	59	726584	5213489	
	+ AFB	2164	0	**669970**	4841286	7.10
	Dragon	9536	0	674494	**4466114**	
	QPlace	2426	0	724930	5338340	

Table 3: 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 represents the total (placement + routing) CPU time in seconds. For routability results, we report the 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 WRoute 2.4.

wirelength reductions for the hard and easy instances. In addition, the proposed approach improves routability, the routing runtime, and the number of vias.

Acknowledgments: The authors would like to thank Igor L. Markov from the University of Michigan for the useful discussions and for suggesting a number of ideas for improving feedback runtime.

6. REFERENCES

[1] S. Adya, I. L. Markov and P. Villarrubia, "On Whitespace and Stability in Mixed-Size Placement," in *Proc. IEEE International Conference on Computer Aided Design*, 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 International Conference on Computer Aided Design*, 2003, pp. 307–310.

[3] C. J. Alpert, J. H. Huang and A. B. Kahng, "Multilevel Circuit Partitioning," in *Proc. ACM/IEEE Design Automation Conference*, 1997, pp. 530–533.

[4] A. Caldwell, A. Kahng and I. Markov, "End-Case Placers for Standard-Cell Layout," in *Proc. ACM/IEEE International Symposium on Physical Design*, 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 Conference*, 2000, pp. 477–482.

[6] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Optimal Partitioners and End-case Placers for Standard-cell Layout," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19(11), 2000, pp. 1304–1313.

[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 Conference*, 2003, pp. 621–627.

[8] R. Dorf and R. Bishop, *Modern Control Systems*, 9th ed., Prentice Hall, 2000.

[9] A. E. Dunlop and B. W. Kernighan, "A Procedure for Placement of Standard-Cell VLSI Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 4(1), 1985, pp. 92–98.

[10] http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/Capo, *GSRC Bookshelf*.

[11] D. J.-H. Huang and A. B. Kahng, "Partitioning-Based Standard-Cell Global Placement With an Exact Objective," in *Proc. ACM/IEEE International Symposium on Physical Design*, 1997, pp. 18–25.

[12] I. L. Markov, Private communication, November 2003.

[13] P. R. Suaris and G. Kedem, "A Quadrisection-Based Combined Place and Route Scheme for Standard Cells," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 8(3), 1989, pp. 234–244.

[14] M. Wang, X. Yang and M. Sarrafzadeh, "DRAGON2000: Standard-Cell Placement Tool for Large Industry Circuits," in *IEEE Proc. International Conference on Computer Aided Design*, 2001, pp. 260–263.

[15] X. Yang, B. Choi and M. Sarrafzadeh, "Routability Driven White Space Allocation for Fixed-Die Standard-Cell Placement," in *Proc. ACM/IEEE International Symposium on Physical Design*, 2002, pp. 42–47.

[16] M. Yildiz and P. Madden, "Improved Cut Sequences for Partitioning Based Placement," in *Proc. ACM/IEEE Design Automation Conference*, 2001, pp. 776–779.

[17] M. Yildiz and P. Madden, "Global Objectives for Standard-Cell Placement," in *Proc. IEEE Great Lakes Symposium on VLSI*, 2001, pp. 68–72.

[18] K. Zhong and S. Dutt, "Effective Partition-Driven Placement With Simultaneous Level Processing and Global Net Views," in *Proc. IEEE International Conference on Computer Aided Design*, 2000, pp. 171–176.