Routability-Driven Placement and White Space Allocation

Chen Li*, Min Xie†, Cheng-Kok Koh*, Jason Cong†, Patrick H. Madden‡
*ECE Purdue University, †CSD UCLA, ‡CSD SUNY Binghamton
{li35,chengkok}@ecn.purdue.edu {xie,cong}@cs.ucla.edu pmadden@cs.binghamton.edu

ABSTRACT

In this paper, we present a congestion-driven placement flow. First, we consider in the global placement stage the routing demand to re-place cells in order to avoid congested regions. Then we allocate appropriate amounts of white space into different regions of the chip according to the congestion map. Finally, a detailed placer is applied to legalize placements while preserving the distributions of white space. Experimental results show that our placement flow can achieve the best routability with the shortest routed wirelength among all publicly available placement tools. Moreover, our white space allocation approach can significantly improve the routabilities of placements generated by other placement tools.

1. INTRODUCTION

Minimizing half-perimeter wirelength is the most typical objective of a placement tool. However, a placement with shorter half-perimeter wirelength (HPWL) may still be unroutable because the routing resources and routing demands at some parts of the chip are not matched. Routability can be optimized by either reducing routing demands or increasing routing resources at congested regions in a placement.

Reducing the routing demands in a congested region is usually performed in the global placement stage, as the cell locations are adjusted only slightly in the detailed placement step. In addition, net topology manipulation can also be considered for optimization, so that good routability can be obtained without much increase in wirelength cost. In [16], the congestion of a placement bin, which is estimated by Rent rule implicitly, is incorporated into the half-perimeter wirelength cost function in a simulated annealing flow. The approach reserves routing resources for global nets by avoiding excessive usage by local nets. However, the congestion contributed by internal nets within a bin is ignored; hence, the accuracy of the congestion estimation is reduced. Furthermore, the topology of the global nets are also ignored. [21] uses a post-processing step of moving cells with Steiner tree reconstructions during placement. The movement of a cell is limited, and reconstructing a Steiner tree for each cell movement is still expensive. The mPG placer [10], which follows a multi-level simulated annealing flow, incorporates into its cost function routing congestion estimated by a tree-based global router. It reduces the routing overflow by 50%. However, the runtime is increased by at least 5 times due to the high complexity involved in the construction of routing trees.

Orthogonal to routing demand reduction is the approach that increases routing resources in congested regions. In fixed-die placement, where white space is typically present, allocating white space to increase routing resources in congested regions is a common method to relieve congestion.[1] White space allocation can be done both during the global placement stage (or at the end of global placement), or during the detailed placement stage. Depending on the specific stage, the congestion estimation method as well as the bin granularity may differ. Allocating white space to congested region can be achieved by inflating cells [5, 15]. In BonnPlace [5], congestions of initial partitions are first estimated by taking both inter-region nets and intra-region nets into account. Congestion due to inter-region nets is estimated by a probabilistic method using a routing grid structure, and congestion due to intra-region nets is estimated by pin density within this region. After that, white space is allocated by expanding cell areas in the congested regions. In [20], congestion in the horizontal or vertical direction is relieved by expanding the region in that direction and shrinking the region in the orthogonal direction in a quadratic placement framework. In [24], expansion of congested regions is formulated as an integer programming problem. Congestion-driven Dragon [23] allocates white space in two steps. White space is first distributed into rows and then into bins within a row. As that may increase the row imbalance, Dragon imposes a lower bound and an upper bound on the amount of white space available in a row.

In this paper, we present a routability-driven placement flow. We propose a congestion-driven multilevel global placement method that enhances the routability during global placement by re-placing cells to avoid congested regions. Experiments showed that compared with its wirelength driven mode, it significantly reduces the global routing overflow and enhances the detailed routing completion rate. Final routed wirelengths are also reduced. We also propose a congestion-driven white space allocation method that after global placement stage, allocates white space to provide appropriate routing resources to congested regions without a great perturbation on wirelength. Experiments show that it can significantly improve routability of placements generated by various latest placement tools. Combining the proposed routability-driven global placement and white space allocation methods, we achieve placements with the best routability among the publicly available placement tools, with all IBM-Dragon version 2 easy and hard benchmark circuits [23] successfully routed.

2. CONGESTION ESTIMATION

As is summarized in [13], existing congestion estimation methods can be divided into two categories: topology-based methods (TP-based), where routing trees are explicitly constructed on some routing grid, and topology-free methods (TP-free), where no explicit routing is needed.

Among the two categories, TP-free modeling is usually faster. This category includes bounding-box (BBOX)-based modeling [12], probabilistic analysis-based modeling [18, 15], Rent's rule-based modeling [25], and pin density-based modeling [5]. TP-based modeling methods usually construct Steiner tree for each net in the netlist. Such modeling method can generate an upper bound for

[1]Evenly distributed or randomly distributed white space may increase the wirelength. White space management methods in [4] and [1] avoid that. However, as these methods are not guided by congestion information, they may not be effective in reducing congestion.

the routability estimation. If the topology generated by a TP-based modeling method is similar to what the after-placement-router does, high fidelity of the modeling can be expected. The Steiner trees can be either precomputed [19], or constructed dynamically [10]. In this work, we take the TP-based approach developed in [10] and construct spanning tree based routing topology to estimate congestion.

2.1 Routing Resource Estimation

In our workflow, we calculate the resource in a routing region as

$$RR_r = \sum_{i=1}^{n} \frac{A_i}{w_i + s_i},$$

where n is the number of routing layers, w_i and s_i are width and space of metal wires in layer i, respectively. A_i denotes the area of layer i available for routing over the region r. In particular, layer 1 and layer 2 might be utilized for routing within the cells; thus, part of these two layers might not be available for signal routing. Layer 3 and above are typically available for signal routing.

2.2 Routing Demand Estimation

To estimate the congestion of a placement, we build a routing grid of $m \times n$ over the chip on each level. As for routing demand estimation, the most accurate value usually comes from global routing itself. However, due to its complexity, global routing cannot be performed very frequently, whereas the placement can go through many changes in each stage, changing the routing congestion at the same time. The accuracy of global routing degrades with the progress of the placement refinement. Therefore, full-blown global routing at every step is both expensive and unnecessary. A congestion estimator with good fidelity and high sensitivity to placement changes is better suited for our purpose. In our framework, we start from a minimum spanning tree for each net, and decompose multi-pin nets into two-pin connections. Then we use the two-bend LZ router developed in [10] to determine the topology for each two-pin connection. This router uses auxiliary data structures to find good quality routes by performing a binary search of the possible routes for a two-pin net. The accuracy of this model is confirmed by a previous study [22] that shows that a large portion of the nets in the netlist are routed using L or Z shaped topology. Consequently, it can be used to guide each step during placement.

Once the topology for each net is determined, the routing demand by net_k on grid cell gc_{ij} is calculated as:

$$RD_{net_k,gc_{ij}} = \begin{cases} w_{net_k} & \text{if } net_k \text{ crosses } gc_{ij}; \\ \alpha \times pin_{net_k} \times w_{net_k} & \text{if } net_k \text{ is within } gc_{ij}; \\ 0 & \text{otherwise.} \end{cases}$$

Here, w_{net_k} is the wire width of net_k, α is a user specified constant, and pin_{net_k} is the number of pins in net_k. (We use $\alpha = 0.25$ in our experiments.) The routing demand on grid cell gc_{ij} and the average routing demand over all grid cells are calculated respectively as:

$$RD_{gc_{ij}} = \sum_{\text{for all } net_k} RD_{net_k,gc_{ij}}, \quad \overline{RD}_{gc} = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} RD_{gc_{ij}}}{mn}.$$

We compute the routing demand for a rectangular region r as:

$$RD_r = \sum_{gc_{ij} \in r} RD_{gc_{ij}}.$$

We define the overflow on a grid cell gc_{ij} and a region r as:

$$OVL_{gc_{ij}} = max\left(0, RD_{gc_{ij}} - \eta RR_{gc_{ij}}\right),$$
$$OVL_r = max\left(0, RD_r - \eta RR_r\right),$$

where η is an empirical multiplier. We determine η in a region r as follows:

$$\eta = \frac{k_r \overline{RD}_{gc}}{RR_r},$$

where k_r is the number of grid cells contained in region r.

We define the overflow caused by net_k as:

$$OVL_{net_k} = \sum_{\{gc_{ij} | OVL_{gc_{ij}} > 0\}} RD_{net_k,gc_{ij}}.$$

The overflow of a placement P is defined as:

$$OVL_P = \sum_{i=1}^{m}\sum_{i=1}^{n} OVL_{gc_{ij}}.$$

Note that according to our definition, the sum of OVL_{net_k} over all nets can be greater than OVL_P due to double counting.

3. ROUTABILITY CONTROL IN GLOBAL PLACEMENT

The global placement in our workflow is built upon mPL [8, 9], a multilevel standard cell placement engine. Fig. 1 shows its flow, including a coarsening phase in which cells are recursively aggregated into clusters, an initial placement generation at the coarsest level, and a refinement phase that refines each coarser level placement solution to get a finer level solution.

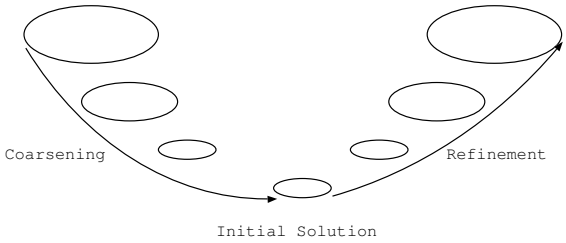

Figure 1: Multilevel global placement. It consists of a coarsening phase, a initial solution generation, and a refinement phase.

Compared with other state-of-the-art placement tools, mPL produces competitive placement results in terms of HPWL. However, mPL gives little consideration for routing congestion. The placement it produces may be overly congested for subsequent routing. Since refinement is the stage where the cell locations as well as the net topologies are determined, our focus for routability control is mainly on the refinement on each level.

The congestion driven refinement during global placement begins with a congestion estimation using a fast LZ router. This is followed by a normal wirelength minimization step. In the end, a subset of cells is chosen and re-placed to adjust the topology of the nets incident on them, so that the routing demand for current placement can be reduced.

To reduce the routing demand, we selectively re-place a subset of the cells after the wirelength minimization step, so that the topology of the nets incident on them can be adjusted. A secondary objective during this process is still to reduce the wirelength. Migrating cells for routability enhancement has been proposed in [16, 10]. However, in [16], the focus is mainly on local net routing demand, and

Algorithm: Congestion Driven Cell Re-placement

Sort nets in descending order of overflow

Pick the first s nets such that $\sum_{i=1}^{s} OVL_{net_i} \geq OVL_P$

for i = 1 to s **do**
  for each cell c connected by net_i **do**
    $WL_{min} = \infty$
    Determine grid cell g_{ij} for c's optimal HPWL
    for each neighboring grid cell $gc_{i'j'}$ of g **do**
      Place c in $gc_{i'j'}$
      Re-determine the topology of the nets incident on c
      Evaluate WL_c using (1)
      if $WL_{min} > WL_c$ **do**
        $gc_{min} = gc_{i'j'}$
        $WL_{min} = WL_c$
      end if
    end for
    Place c in gc_{min}
  end for
end for

Figure 2: Algorithm for congestion driven cell re-placement.

the routing topology of global nets are ignored. The approach in [10] is quite indiscriminate about which cells should be re-placed. As a consequence, there are many candidate cells for re-placement, resulting in prohibitive runtime. In our workflow, candidates for cell re-placement are picked based on the routing topology of nets incident on them.

Fig. 2 gives a description of this process. We sort the nets according to the overflow they cause in descending order, and pick the first s nets, such that the sum of their overflow is more than the total overflow of the current placement. The cells connected by these nets will be re-placed.

For a cell c, we first determine the grid cell gc_{ij} corresponding to its optimal location for half-perimeter wirelength. Then we try to place c in each of the neighboring grid cells within a certain distance d, i.e., $\{gc_{ij} \mid |i'-i| + |j'-j| \leq d\}$. After c is placed in a grid cell, the topology for the nets incident on it is re-determined using the LZ router. The new placement for c is evaluated using a weighted wirelength of all the nets incident on c:

$$WL_c = \sum WGT_{net_k} \times WL_{net_k},$$

where WGT_{net_k} is the weight on net_k, calculated as the average congestion of the grids cell net_k crosses, and WL_{net_k} is the half perimeter wirelength of net_k. In the end, c is placed in the region that results in the shortest weighted wirelength. Fig. 3 shows an example of this process. The legend corresponds to the congestion in different routing regions. Starting from the optimal location for half-perimeter wirelength, we search the neighborhood for a region that gives the shortest weighted wirelength. In this example, the optimal location for the half-perimeter wirelength gives a weighted wirelength of 8.8 due to the congestion on each route, whereas a neighboring region will give a smaller weighted wirelength of 6.2.

The total cell area in a routing region might exceed its capacity after this procedure. To slightly relieve this, we lock the recently replaced cells and try to rebalance the area density with ripple move. This re-placement process is repeated for several passes before going to the next level.

4. WHITE SPACE ALLOCATION

In our global placement flow, the amount of white space in a region may not accurately match its routing demand. Therefore, we further apply a white space allocation step to allocate appropriate amounts of white space into congested regions. In the context of fixed-die placement, this step does not increase the chip area as white space is already present .

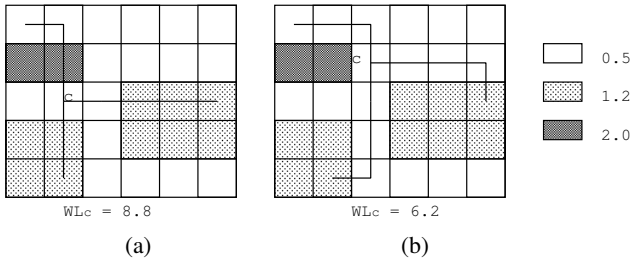

Figure 3: Congestion driven cell re-placement. The legend corresponds to the congestion in different routing regions. (a) Original placement of cell c in the optimal location for half-perimeter wirelength gives a weighted wirelength of 8.8. (b) Re-placement of c in a neighboring region gives a weighted wirelength of 6.2.

Unlike Dragon's two-step allocation approach [23], we assign appropriate amounts of white space to congested regions in a hierarchical flow. We first construct a slicing tree based on the geometric locations of all cells. We estimate the congestion level at each node of the tree and then adjust the cut line location at each node in a top-bottom fashion to distribute the white space to two child nodes. After white space allocation, we apply a detailed placement to remove overlaps and further reduce HPWL while preserving the white space distribution. In the rest of this section, we describe this flow in details.

4.1 Slicing Tree and Congestion Estimation

Given any global or detailed placement, we first construct a slicing tree using a method that is similar to a partitioning-based global placement flow. The difference is that the partitioning here is performed based on the geometric locations of the cells (instead of the minimization of cut size).

We recursively partition the placement, starting from the full chip level until every region contains a small number of cells. Cut directions are determined by comparing the aspect ratio of this region with a fixed value. Each cut line geometrically bisects a region evenly. For a region, once its cut direction and cut location are determined, all cells whose centers are located at the left of the cut line (if cut vertically) or above the cut line (if cut horizontally) form the left child of that tree node. The remaining cells in the region form the right child of that node. This is essentially the slicing tree data structure used to capture a slicing floorplan (as shown in Fig. 4(a)). Every node in the tree maintains its cut direction, cut location, congestion, total cell area as well as cell list.

Now, we estimate the congestion level of the nodes in the slicing tree in a bottom-up fashion. The congestion level of a leaf node can be estimated by the total routing overflow of the grid cells contained in this leaf node. The congestion level of an internal node can then be computed through a post-order traversal of the tree by adding up the congestion levels of two child nodes.

4.2 Cut Line Adjustment

After performing congestion analysis on all tree nodes, we shift cut line locations in the nodes of the slicing tree by traversing the nodes in a top-down fashion such that the amounts of white space allocated to the two child nodes are linearly proportional to

their congestion levels. Consider a region r with lower-left corner (x_0, y_0), upper-right corner (x_1, y_1) and the original vertical cut direction at $x_{cut} = (x_0 + x_1)/2$. Thus, the area of this region is $A_r = (x_1 - x_0)(y_1 - y_0)$. Assume that the total area of cells for left subregion r_0 and right subregion r_1 are S_0 and S_1, and the corresponding congestion levels are OVL_0 and OVL_1, respectively. We want to distribute the total amount of white space, which is $(A_r - S_0 - S_1)$, to the two subregions such that the amounts of white space in the two subregions are linearly proportional to their congestion levels. Thus, the amount of white space allocated to subregion r_0 is $(A_r - S_0 - S_1)\frac{OVL_0}{OVL_0+OVL_1}$. Then, the new cut line location x'_{cut} can be derived as follows:

$$\gamma = \frac{S_0 + (A_r - S_0 - S_1)\frac{OVL_0}{OVL_0+OVL_1}}{A_r},$$
$$x'_{cut} = \gamma x_1 + (1-\gamma)x_0,$$

where γ is the ratio of the left subregion area to A_r after the cut line adjustment.

This step is similar to a top-down partitioning-based global placement except that the cut direction, the cut location and sub-netlists are all known. We also constrain all cells to stay at the center of the region to which they belong during this top-down flow. After this step, we obtain a global placement that contains overlaps. Moreover, cells may not be placed along a row. Note that the cut line adjustment approach is performed in the same spirit as fractional cut [2], where horizontal cuts are not aligned with row boundaries. We illustrate the region changes before and after cut line adjustment by an example in Fig. 4(b). In this example, we show the total cell area and congestion level at every tree node of the slicing tree. Cut lines are adjusted from top to bottom such that white spaces in the subregions are proportional to their congestion levels.

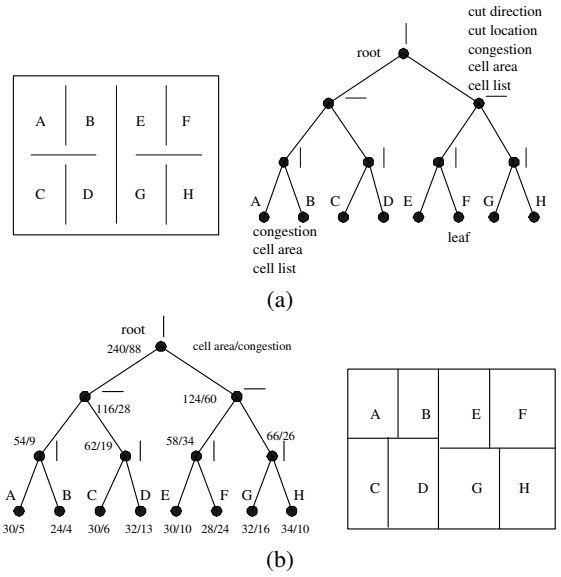

Figure 4: (a) A slicing tree and its corresponding cut lines and regions. (b) A slicing tree after congestion estimation and regions after cut lines adjustment.

4.3 Detailed Placement

After white space allocation, the cells may overlap and they may not be placed along rows. We need a detailed placer to legalize the placement. This detailed placer should be able to maintain the locations of white spaces such that the white spaces will not be greatly redistributed. As the detailed placer DOMINO [14] generates packed placements to minimize half-perimeter wirelength, it does not fit our purpose.

We propose a detailed placer to preserve white space distribution and further reduce HPWL. We first adopt a greedy legalization algorithm [17] to remove the overlaps, then we locally minimize HPWL using a sliding window approach [7, 2]. We slide a single-row or double-row window across the entire chip, and perform cell swapping within the window. White space inside the window is treated as dynamic-width pseudo-cells, whose widths may shrink or expand, depending on the need imposed by cell swapping. As a result, more cell swapping can be considered within a window, leading to a better HPWL.

5. EXPERIMENTAL RESULTS

In the following, we use **mPL-R** to denote our global placement flow, **WSA** our white space allocation flow, and **mPL-R+WSA** the combined flow.

We evaluate the effectiveness of our tools (**mPL-R**, **WSA** and **mPL-R+WSA**) in achieving routability by comparing them with several state-of-the-art academic tools, including **Dragon** 3.01 [23], **CAPO** 8.8 [6], **Feng Shui** 2.2 [2], and **mPG** 1.0 [11], a leading-edge industrial placement tools **Cadence QPLACE** (in SEULTRA 5.3). **Dragon** is run with -fd option for congestion-driven mode. **CAPO** is run with rowironing turned off. **mPG** is run using the wirelength objective instead of the congestion minimization objective due to the huge runtime of congestion-driven **mPG**. As **mPG** generates global placements with overlaps, we apply **QPLACE ECO** mode to obtain the final placements.

All experiments are performed on the complete set of IBM-Dragon version 2 easy and hard benchmarks. These benchmarks were converted from ISPD98 [3] by mapping cells to commercial standard cell library by authors of **Dragon** [23] to evaluate routability of placements. The relative amounts of white space in these benchmarks are shown in Table 1.

All placement tools are run five times for each benchmark with the exception that our tools and **QPLACE** are run only once for each benchmark as different runs of our tools and **QPLACE** generate the same placement. **Dragon**, **CAPO**, and **Feng Shui** are run on a Pentium4 2.6GHz CPU with 512MB memory. **mPG**, **Cadence QPLACE**, and **WROUTE** are run on an UltraSpacII 450MHz CPU with 1GB memory.

The results are given in Table 1–4. All data are obtained by averaging over the runs except the column "S/V/F". In Table 1, 3, and 4, the columns "r-WL", "vias", "vlts", "o.c.%" and "r-time" show the routed wirelengths, numbers of vias, numbers of violations, percentages of over-capacity gcells and routing times reported by **WROUTE**, respectively.

The column "S/V/F" shows the total number of different status of routing results for five runs or one run. The status of a routing result can be one of the following: successful routing without violation (denoted as 'S'), finished routing with some violations (denoted as 'V'), and failed routing due to too many violations or too long a routing time (denoted as 'F'). For example, "1/0/4" means that there are 1 successful routing, 0 finished routing, and 4 failed routings in five runs. The column "r-time" shows the average for only successful or finished routings. Failed routings are excluded in the routing time comparison, as they have either a very short routing time or a time of 24 hours (the runtime limit for the router), which if included would skew the comparison. The row "summary" summarizes the results of these placement tools on this

suite of benchmarks. The column "S/V/F" in this row shows the total numbers of successful routings, finished routings and failed routings on easy and hard benchmarks. The column "vlts" shows the average numbers of violations of all easy and hard benchmarks.

5.1 Routability Comparison

From Table 1, we observe that our placement flow of **mPL-R+WSA** obtains successful routings for all 16 benchmarks. whereas other tools can only have partial successful routing results. For example, for 5 runs of Dragon and CAPO on each of the 16 benchmarks, **Dragon** obtains 52 successful, 18 finished and 10 failed routings, and **CAPO** obtains 25 successful, 24 finished and 31 failed routings. For one run on each of the 16 benchmarks, **QPLACE** obtains 12 successful and 4 finished routings. Moreover, among all tools, our placement flow obtains the smallest routed wirelengths, whereas those of **Dragon** and **QPLACE** are 8.7% and 12.5% longer. We also obtain fewer vias and fewer over-capacity gcells. We conclude that our placement flow of **mPL-R+WSA** is the best in terms of routability and routed wirelength.

The runtime of various placement tools including **mPL-R** and **WSA** on these benchmarks are shown in Table 2. In the row "summary", we show the ratios of runtimes among these tools with respect to that of our flow on both easy and hard benchmarks. Among these tools, **Dragon**, **QPLACE**, and **mPL-R+WSA** achieve better routability on these IBM benchmarks. Among these three tools, **QPLACE** is most scalable in terms of runtime and **Dragon** is the least scalable.

5.2 Impacts of Various Routability Optimization Techniques

Table 3 compares the impacts of each technique in our workflow. A summary is given at the end of table. It can be seen that both routability control technique during global placement stage and white space allocation technique after global placement stage are effective in relieving routing congestion. Compared to **mPL**, these two techniques can reduce the overflowed global routing cells by 83% and 72%, and improve the completion rate from zero successful routings to 14 and 9 successful routings out of 16 routings, respectively. The combined flow has successful routings on *all* these benchmarks. Routed wirelengths are improved significantly by 11.6%. Overall, the combined workflow is the best in terms of final completion rate and routed wirelength.

5.3 Impacts of White Space Allocation

We further evaluate the impacts of the white space allocation technique by applying it on the placements generated by other tools. Routability results are shown in Table 4.

Table 4 shows that our approach can greatly improve the routability of a placement, especially on those placements generated by wirelength-driven placement tools. After applying our approach, **Dragon** obtains 52 successful, 26 finished and 2 failed routing results and **CAPO** obtains 48 successful and 27 finished and 5 failed routing results; similar results are obtained for **Feng Shui**, **mPG** and **QPLACE**. Compared to results obtained by these tools without our white space allocation technique (see the summary row in Table 4), we consistently reduce the routed wirelength by 1.1% to 8.0%. We also reduce the number of vias, violations and significantly reduce the over-capacity gcells (with the exception of **QPLACE**). We conclude that routability and routed wirelength can be *simultaneously* improved with our white space allocation technique.

6. CONCLUSION

We propose a congestion-driven global placement that enhances the routability by considering routing resource demand reduction. We also propose a congestion-driven white space allocation approach can further allocate appropriate amounts of white space into congestion regions to increase routing resources. Experimental results show that our placement flow can greatly improve routability. We achieve successful routings on all IBM easy and hard benchmark circuits with the best routed wirelength and competitive runtime.

7. ACKNOWLEDGMENTS

This research is partially supported by the Semiconductor Research Corporation under Contract 2003-TJ-1091 and Project 947.1, National Science Foundation under Awards CCR-0096383 and CCR-9984553, and an IBM Faculty Partnership Award.

8. REFERENCES

[1] S. Adya, I. Markov, and P. Villarrubia. On whitespace and stability in mixed-size placement and physical synthesis. In *Proc. Int. Conf. on Computer Aided Design*, pages 311–318, 2003.

[2] A. Agnihotri, M. C. Yildiz, A. Khatkhate, A. Mathur, S. Ono, and P. H. Madden. Fractional cut: Improved recursive bisection placement. In *Proc. Int. Conf. on Computer Aided Design*, pages 307–310, 2003.

[3] C. J. Alpert. The ISPD98 circuit benchmark suite. In *Proc. Int. Symp. on Physical Design*, pages 80–85, 1998.

[4] C. J. Alpert, G.-J. Nam, and P. G. Villarrubia. Effective free space management for cut-based placement via analytical constraint generation. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 22(10):1343–1353, 2003.

[5] U. Brenner and A. Rohe. An effective congestion driven placement framework. In *Proc. Int. Symp. on Physical Design*, pages 6–11, 2002.

[6] A. E. Caldwell, A. B. Kahng, and I. L. Markov. Can recursive bisection alone produce routable placements? In *Proc. Design Automation Conf.*, pages 477–482, 2000.

[7] A. E. Caldwell, A. B. Kahng, and I. L. Markov. Optimal partitioners and end-case placers for standard-cell layout. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 19(11):1304–1313, 2000.

[8] T. Chan, J. Cong, T. Kong, and J. Shinnerl. Multilevel optimization for large-scale circuit placement. In *Proc. Int. Conf. on Computer Aided Design*, pages 171–176, 2000.

[9] T. Chan, J. Cong, T. Kong, J. Shinnerl, and K. Sze. An enhanced multilevel algorithm for circuit placement. In *Proc. Int. Conf. on Computer Aided Design*, pages 299–306, 2003.

[10] C.-C. Chang, J. Cong, Z. Pan, and X. Yuan. Physical hierarchy generation with routing congestion control. In *Proc. Int. Symp. on Physical Design*, pages 36–41, 2002.

[11] C.-C. Chang, J. Cong, Z. Pan, and X. Yuan. Multilevel global placement with congestion control. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 22(4):395–409, 2003.

[12] C.-L. E. Cheng. RISA: accurate and efficient placement routability modeling. In *Proc. Int. Conf. on Computer Aided Design*, pages 690–695, Nov. 1994.

[13] J. Cong, T. Kong, J. Shinnerl, M. Xie, and X. Yuan. Large-scale circuit placement: Gap and promise. In *Proc. Int. Conf. on Computer Aided Design*, pages 883–890, November 2003.

Table 1: Routability comparison of our tool with various placement tools. All wirelengths are scaled by 10^8.

bench-marks	tools	routed by Cadence WROUTE						bench-marks	tools	routed by Cadence WROUTE					
		S/V/F	r-WL	vias	vlts	o.c.%	r-time			S/V/F	r-WL	vias	vlts	o.c.%	r-time
ibm01	Dragon-fd	1/0/4	0.929	146773	5346	5.64	1:26:47	ibm01	Dragon-fd	0/0/5	0.904	145770	6998	6.63	N/A
-easy	CAPO	0/0/5	0.887	140529	9397	6.61	N/A	-hard	CAPO	0/0/5	0.899	142496	11594	8.26	N/A
	FS	0/0/5	0.852	142195	21295	10.01	N/A		FS	0/0/5	0.870	142812	21900	10.70	N/A
14.88%	mPG+ECO	0/0/5	1.003	147230	23225	10.40	N/A	12.00%	mPG+ECO	0/0/5	1.001	148382	29628	11.95	N/A
	QPLACE	1/0/0	0.875	132250	0	3.21	0:28:01		QPLACE	0/1/0	0.831	133469	20	3.90	1:18:17
	mPL-R+WSA	1/0/0	0.772	127969	0	1.57	0:28:37		mPL-R+WSA	1/0/0	0.751	129648	0	2.38	0:38:57
ibm02	Dragon-fd	5/0/0	2.18	313289	0	1.07	1:17:19	ibm02	Dragon-fd	2/2/1	2.24	321523	2045	2.59	2:48:18
-easy	CAPO	1/1/3	2.41	321415	9979	5.06	4:55:39	-hard	CAPO	0/0/5	2.45	328943	24031	8.42	N/A
	FS	0/0/5	2.37	321705	15440	4.93	N/A		FS	0/0/5	2.40	325003	19036	6.29	N/A
9.58%	mPG+ECO	0/0/5	2.55	338704	31894	9.14	N/A	4.72%	mPG+ECO	0/0/5	2.59	349984	58670	13.89	N/A
	QPLACE	1/0/0	2.11	289985	0	0.27	0:31:02		QPLACE	0/1/0	2.20	317981	1	1.59	1:49:20
	mPL-R+WSA	1/0/0	1.89	284396	0	0.43	0:40:44		mPL-R+WSA	1/0/0	1.94	296290	0	0.97	0:58:21
ibm07	Dragon-fd	4/1/0	4.55	593637	4.4	0.68	1:46:48	ibm07	Dragon-fd	4/1/0	4.77	624005	0.8	2.51	3:34:02
-easy	CAPO	0/4/1	4.93	614172	186	3.19	8:59:29	-hard	CAPO	0/0/5	5.36	642462	33980	7.78	N/A
	FS	0/2/3	4.49	617027	1042	3.65	22:00:27		FS	0/1/4	4.58	611699	6193	4.12	16:35:49
10.05%	mPG+ECO	0/0/5	5.67	666646	83489	10.91	N/A	4.70%	mPG+ECO	0/0/5	5.54	677474	113178	12.29	N/A
	QPLACE	0/1/0	4.67	561838	2	0.31	1:16:59		QPLACE	1/0/0	5.07	607972	0	2.17	2:43:00
	mPL-R+WSA	1/0/0	4.29	548765	0	0.35	1:28:13		mPL-R+WSA	1/0/0	4.43	579157	0	1.71	2:22:12
ibm08	Dragon-fd	5/0/0	4.78	696404	0	0.08	0:59:25	ibm08	Dragon-fd	3/2/0	4.71	721215	0.4	0.28	2:01:41
-easy	CAPO	2/3/0	5.16	724992	30.6	0.98	7:00:34	-hard	CAPO	0/1/4	5.59	782283	22841	3.97	18:17:42
	FS	0/3/2	5.19	767378	12947	3.31	18:53:31		FS	0/4/1	5.08	760494	248	2.41	13:01:35
9.97%	mPG+ECO	0/1/4	5.56	784389	23700	4.28	13:17:23	4.84%	mPG+ECO	0/0/5	5.50	788772	45339	5.95	N/A
	QPLACE	1/0/0	5.32	706212	0	0.37	1:27:59		QPLACE	1/0/0	5.18	713999	0	0.28	1:44:54
	mPL-R+WSA	1/0/0	4.58	661733	0	0.06	0:59:52		mPL-R+WSA	1/0/0	4.49	684910	0	0.24	1:28:55
ibm09	Dragon-fd	5/0/0	3.81	594621	0	0.03	0:53:27	ibm09	Dragon-fd	5/0/0	3.70	603149	0	0.04	0:58:36
-easy	CAPO	4/1/0	3.77	581190	0	0.04	0:56:30	-hard	CAPO	4/1/0	3.84	602751	0.2	0.09	1:04:06
	FS	3/2/0	3.56	586891	0.4	0.06	1:01:49		FS	5/0/0	3.66	593298	0	0.10	1:06:24
9.76%	mPG+ECO	5/0/0	4.00	616150	0	0.21	1:16:47	4.88%	mPG+ECO	4/1/0	4.28	653824	2.2	0.86	2:21:01
	QPLACE	1/0/0	4.04	561859	0	0.01	47:27		QPLACE	1/0/0	3.90	578085	0	0.03	1:03:50
	mPL-R+WSA	1/0/0	3.50	549568	0	0.02	0:50:32		mPL-R+WSA	1/0/0	3.65	570032	0	0.02	0:59:52
ibm10	Dragon-fd	3/2/0	7.46	934786	7.2	0.04	1:53:43	ibm10	Dragon-fd	5/0/0	7.16	939199	0	0.08	1:43:38
-easy	CAPO	4/1/0	7.54	934359	7.6	0.19	3:27:54	-hard	CAPO	2/3/0	7.87	990705	0.8	0.66	5:13:37
	FS	4/1/0	7.02	937774	9.6	0.22	3:29:00		FS	1/4/0	7.02	931323	24.8	0.17	4:17:50
9.78%	mPG+ECO	3/1/1	7.91	990337	376	1.03	4:16:29	4.92%	mPG+ECO	1/1/3	8.26	1051147	8752	2.27	22:09:05
	QPLACE	1/0/0	7.32	877598	0	0.02	1:13:10		QPLACE	1/0/0	7.47	916207	0	0.06	2:01:55
	mPL-R+WSA	1/0/0	6.84	873311	0	0.02	1:28:06		mPL-R+WSA	1/0/0	6.76	902026	0	0.06	1:57:34
ibm11	Dragon-fd	2/3/0	5.68	780344	0.8	0.05	1:09:08	ibm11	Dragon-fd	4/1/0	5.57	795088	0.2	0.17	1:29:21
-easy	CAPO	4/1/0	5.65	767643	0.2	0.21	1:34:16	-hard	CAPO	3/2/0	5.85	811186	0.6	0.85	2:24:01
	FS	3/2/0	5.41	774667	0.4	0.23	1:44:42		FS	5/0/0	5.43	770152	0	0.28	1:42:40
9.89%	mPG+ECO	3/2/0	5.93	813960	0.4	0.89	2:36:53	4.67%	mPG+ECO	0/4/1	6.54	902662	6958	3.70	12:55:47
	QPLACE	1/0/0	6.05	745184	0	0.08	1:04:44		QPLACE	1/0/0	6.10	779713	0	0.22	1:50:30
	mPL-R+WSA	1/0/0	5.16	714824	0	0.03	1:02:26		mPL-R+WSA	1/0/0	5.15	745015	0	0.18	1:41:09
ibm12	Dragon-fd	3/2/0	10.61	1141524	0.4	0.22	2:36:56	ibm12	Dragon-fd	1/4/0	10.50	1162650	0.8	0.48	3:44:46
-easy	CAPO	1/4/0	10.99	1172164	51.4	1.35	11:56:53	-hard	CAPO	0/2/3	10.99	1231308	374.6	2.69	21:45:52
	FS	0/0/5	10.47	1219168	55275	5.33	N/A		FS	0/1/4	10.56	1234216	56301	6.31	21:15:16
14.78%	mPG+ECO	0/0/5	13.25	1397195	266038	14.63	N/A	9.94%	mPG+ECO	0/0/5	12.95	1437522	357235	18.22	N/A
	QPLACE	0/1/0	11.98	1210849	122	2.40	9:44:26		QPLACE	1/0/0	11.09	1174369	0	1.46	4:48:27
	mPL-R+WSA	1/0/0	10.52	1127925	0	0.34	3:54:03		mPL-R+WSA	1/0/0	10.13	1107551	0	0.40	3:49:26
summary	Dragon-fd	52/18/10	1.087×	1.075×	901	1.80×	1.38×	Except for "vlts", which is the average number of violations per run, results in the summary row are normalized with respect to mPL-R+WSA.							
	CAPO	25/24/31	1.147×	1.094×	7030	7.78×	4.30×								
	FS	21/20/39	1.081×	1.089×	13107	10.6×	5.89×								
	mPG+ECO	16/10/54	1.245×	1.178×	66663	28.8×	3.63×								
	QPLACE	12/4/0	1.125×	1.039×	9.1	2.13×	1.25×								
	mPL-R+WSA	16/0/0	1.000	1.000	0	1.00	1.00								

Table 2: Runtime comparison among various placement tools. The runtimes are in formats of "hr:mm:ss" or "mm:ss".

benchmarks	Dragon-fd	CAPO	FS	mPG	QPLACE	mPL	mPL-R	WSA	benchmarks	Dragon-fd	CAPO	FS	mPG	QPLACE	mPL	mPL-R	WSA
ibm01easy	10:30	0:59	2:32	10:49	2:13	4:16	5:40	0:40	ibm01hard	10:23	0:55	2:48	11:04	2:45	4:28	5:45	0:37
ibm02easy	16:20	2:02	4:51	25:51	4:51	9:32	20:03	1:27	ibm02hard	16:16	1:47	5:29	22:06	6:05	9:55	18:43	1:18
ibm07easy	40:11	5:25	11:46	40:16	10:05	20:48	29:02	2:24	ibm07hard	26:08	7:07	12:52	48:18	12:28	21:52	27:54	2:06
ibm08easy	1:39:40	6:49	14:22	58:37	13:17	28:51	40:44	3:46	ibm08hard	1:04:54	5:57	15:26	1:04:44	17:07	30:24	40:19	3:21
ibm09easy	1:13:38	7:42	13:31	54:57	11:20	24:26	25:22	3:04	ibm09hard	49:57	7:26	14:26	52:42	14:34	24:45	25:31	2:23
ibm10easy	2:05:00	13:00	19:14	1:20:03	17:50	38:24	53:57	5:01	ibm10hard	1:21:34	11:19	21:02	1:23:55	23:05	39:42	50:59	3:54
ibm11easy	1:23:17	13:41	18:48	1:18:19	15:35	30:21	38:52	4:00	ibm11hard	54:12	11:43	20:31	1:10:39	19:02	32:29	37:38	3:07
ibm12easy	2:22:28	13:42	20:25	1:23:06	19:42	43:09	1:19:25	6:24	ibm12hard	1:32:41	12:23	21:25	1:28:52	20:17	44:53	1:16:16	4:50
summary	1.55	0.19	0.37	1.49	0.36	0.67	1.00		Runtimes are normalized with respect to mPL-R+WSA in the summary row.								

Table 3: Impacts of various routability optimization techniques in our flow. All wirelengths are scaled by 10^8.

bench-marks	tools	routed by Cadence WROUTE						bench-marks	tools	routed by Cadence WROUTE					
		S/V/F	r-WL	vias	vlts	o.c.%	r-time			S/V/F	r-WL	vias	vlts	o.c.%	r-time
ibm01	mPL	0/0/1	0.886	143828	16055	7.83	N/A	ibm01	mPL	0/0/1	0.884	143940	16124	8.33	N/A
-easy	mPL-R	1/0/0	0.821	137894	0	3.22	1:07:36	-hard	mPL-R	1/0/0	0.789	137511	0	3.45	1:30:38
	mPL+WSA	1/0/0	0.852	140841	0	3.23	1:56:39		mPL+WSA	1/0/0	0.815	140886	0	4.18	1:38:34
	mPL-R+WSA	1/0/0	0.772	127969	0	1.57	0:28:37		mPL-R+WSA	1/0/0	0.751	129648	0	2.38	0:38:57
ibm02	mPL	0/0/1	2.26	318566	13405	4.66	N/A	ibm02	mPL	0/0/1	2.34	326682	24554	8.50	N/A
-easy	mPL-R	1/0/0	1.96	297454	0	0.42	0:37:40	-hard	mPL-R	0/1/0	2.08	320952	45	2.17	4:31:47
	mPL+WSA	1/0/0	2.06	302274	0	0.80	0:51:10		mPL+WSA	0/0/1	2.22	312616	10721	5.13	N/A
	mPL-R+WSA	1/0/0	1.89	284396	0	0.43	0:40:44		mPL-R+WSA	1/0/0	1.94	296290	0	0.97	0:58:21
ibm07	mPL	0/0/1	4.51	654818	1887	3.29	N/A	ibm07	mPL	0/0/1	5.10	642796	30539	6.85	N/A
-easy	mPL-R	1/0/0	4.35	574541	0	0.34	1:15:28	-hard	mPL-R	1/0/0	4.52	607782	0	1.98	3:18:34
	mPL+WSA	1/0/0	4.17	559221	0	0.24	1:03:24		mPL+WSA	0/1/0	4.45	605580	2	2.41	3:20:19
	mPL-R+WSA	1/0/0	4.29	548765	0	0.35	1:28:13		mPL-R+WSA	1/0/0	4.43	579157	0	1.71	2:22:12
ibm08	mPL	0/1/0	5.13	806862	69	2.39	16:26:36	ibm08	mPL	0/0/1	5.44	791936	47979	6.08	N/A
-easy	mPL-R	1/0/0	4.65	693206	0	0.07	0:57:52	-hard	mPL-R	1/0/0	4.61	717951	0	0.39	2:50:48
	mPL+WSA	0/1/0	4.85	708107	2	0.23	1:42:33		mPL+WSA	0/1/0	5.11	780562	21	2.28	9:54:36
	mPL-R+WSA	1/0/0	4.58	661733	0	0.06	0:59:52		mPL-R+WSA	1/0/0	4.49	684910	0	0.24	1:28:55
ibm09	mPL	0/1/0	3.95	645890	82	0.07	5:14:37	ibm09	mPL	0/1/0	4.00	673932	77	0.17	6:59:18
-easy	mPL-R	1/0/0	3.57	580005	0	0.02	0:49:55	-hard	mPL-R	1/0/0	3.74	601755	0	0.04	0:57:17
	mPL+WSA	1/0/0	3.76	584530	0	0.03	0:57:09		mPL+WSA	1/0/0	3.71	596700	0	0.04	0:55:18
	mPL-R+WSA	1/0/0	3.50	549568	0	0.02	0:50:32		mPL-R+WSA	1/0/0	3.65	570032	0	0.02	0:59:52
ibm10	mPL	0/1/0	6.98	947526	42	0.17	6:48:56	ibm10	mPL	0/1/0	8.05	1071552	55	1.27	14:20:44
-easy	mPL-R	1/0/0	6.94	910034	0	0.02	1:22:47	-hard	mPL-R	1/0/0	6.85	933290	0	0.06	1:53:46
	mPL+WSA	1/0/0	6.80	894979	0	0.03	1:41:30		mPL+WSA	0/1/0	7.47	974989	80	0.30	5:14:43
	mPL-R+WSA	1/0/0	6.84	873311	0	0.02	1:28:06		mPL-R+WSA	1/0/0	6.76	902026	0	0.06	1:57:34
ibm11	mPL	0/1/0	5.62	866097	132	0.40	10:34:03	ibm11	mPL	0/1/0	6.07	918744	91	1.39	10:41:14
-easy	mPL-R	1/0/0	5.25	757252	0	0.03	0:59:50	-hard	mPL-R	0/1/0	5.28	786186	1	0.09	1:25:06
	mPL+WSA	1/0/0	5.29	755682	0	0.05	1:09:15		mPL+WSA	1/0/0	5.34	784769	0	0.22	1:25:01
	mPL-R+WSA	1/0/0	5.16	714824	0	0.03	1:02:26		mPL-R+WSA	1/0/0	5.15	745015	0	0.18	1:41:09
ibm12	mPL	0/0/1	11.16	1208277	2569	2.65	N/A	ibm12	mPL	0/0/1	11.61	1262711	64083	6.07	N/A
-easy	mPL-R	1/0/0	10.48	1155284	0	0.41	4:21:03	-hard	mPL-R	1/0/0	10.27	1151207	0	0.67	4:27:59
	mPL+WSA	0/1/0	10.42	1137643	77	0.39	8:38:46		mPL+WSA	0/1/0	11.12	1229005	82	1.98	18:22:55
	mPL-R+WSA	1/0/0	10.52	1127925	0	0.34	3:54:03		mPL-R+WSA	1/0/0	10.13	1107551	0	0.40	3:49:26
summary	mPL	0/7/9	1.000	1.000	3564	1.00	1.00	Except for "vlts", which is the average number of violations per run, results in the summary row are normalized with respect to mPL.							
	mPL-R	14/2/0	0.906×	0.915×	2.9	0.17×	0.13×								
	mPL+WSA	9/6/1	0.933×	0.926×	687	0.28×	0.18×								
	mPL-R+WSA	16/0/0	0.884×	0.870×	0	0.13×	0.14×								

[14] K. Doll, F. M. Johannes, and K. J. Antreich. Iterative placement improvement by network flow methods. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 13(10):1189–1200, 1994.

[15] W. Hou, H. Yu, X. Hong, Y. Cai, W. Wu, J. Gu, and W. Kao. A new congestion-driven placement algorithm based on cell inflation. In *Proc. Asia South Pacific Design Automation Conf.*, pages 605–608, 2001.

[16] B. Hu and M. Marek-Sadowska. Congestion minimization during placement without estimation. In *Proc. Int. Conf. on Computer Aided Design*, pages 739–745, 2002.

[17] A. Khatkhate, C. Li, A. Agnihotri, M. Yildiz, S. Ono, C.-K. Koh, and P. Madden. Recursive bisection based mixed block placement. In *Proc. Int. Symp. on Physical Design*, pages 84–89, 2004.

[18] J. Lou, S. Krishnamoorthy, and H. Sheng. Estimating routing congestion using probabilistic analysis. In *Proc. Int. Symp. on Physical Design*, pages 112–117, 2001.

[19] S. Mayrhofer and U. Lauther. Congestion-driven placement using a new multi-partitioning heuristic. In *Proc. Int. Conf. on Computer Aided Design*, pages 332–335, 1990.

[20] P. N. Parakh, R. B. Brown, and K. A. Sakallah. Congestion driven quadratic placement. In *Proc. Design Automation Conf.*, pages 275–278, 1998.

[21] R.-S. Tsay, S. Chang, and J. Thorvaldson. Early wirability checking and 2-D congestion-driven circuit placement. In *Proc. Int. Conf. on ASIC*, pages 50–53, 1992.

[22] J. Westra, C. Bartels, and P. Groeneveld. Probabilistic congestion prediction. In *Proc. Int. Symp. on Physical Design*, pages 204–209, 2004.

[23] X. Yang, B.-K. Choi, and M. Sarrafzadeh. Routability-driven white space allocation for fixed-die standard-cell placement. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 22(4):410–419, 2003.

[24] X. Yang, R. Kastner, and M. Sarrafzadeh. Congestion reduction during placement based on integer programming. In *Proc. Int. Conf. on Computer Aided Design*, pages 573–576, 2001.

[25] X. Yang, R. Kastner, and M. Sarrafzadeh. Congestion estimation during top-down placement. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 21(1):72–80, 2002.

Table 4: Routability results after applying our approach WSA to placements generated by various tools. All wirelengths are scaled by 10^8.

bench-marks	tools	routed by Cadence WROUTE						bench-marks	tools	routed by Cadence WROUTE					
		S/V/F	r-WL	vias	vlts	o.c.%	r-time			S/V/F	r-WL	vias	vlts	o.c.%	r-time
ibm01	Dragon-fd+WSA	5/0/0	0.813	137717	0	2.79	1:19:12	ibm01	Dragon-fd+WSA	1/3/1	0.827	140233	1215	4.23	2:00:40
-easy	CAPO+WSA	5/0/0	0.775	127369	0	2.14	0:41:39	-hard	CAPO+WSA	3/2/0	0.787	130241	7.8	3.57	1:28:57
	FS+WSA	4/1/0	0.859	138136	6.8	3.78	1:18:53		FS+WSA	0/0/5	0.870	138896	6505	5.42	N/A
	mPG+WSA	1/0/4	0.903	141236	4866	4.41	1:48:07		mPG+WSA	0/0/5	0.894	141761	9166	6.44	N/A
	QPLACE+WSA	0/1/0	0.848	138959	1	2.76	1:10:09		QPLACE+WSA	1/0/0	0.818	138995	0	3.84	1:29:43
ibm02	Dragon-fd+WSA	3/2/0	2.13	310411	14.2	1.35	2:55:17	ibm02	Dragon-fd+WSA	2/2/1	2.18	322273	2225	2.79	5:20:51
-easy	CAPO+WSA	4/1/0	2.26	303751	0.2	1.77	1:48:40	-hard	CAPO+WSA	1/0/4	2.35	313349	12962	5.82	3:02:48
	FS+WSA	5/0/0	2.24	307941	0	1	1:03:03		FS+WSA	0/0/5	2.34	317854	11127	3.99	N/A
	mPG+WSA	3/1/1	2.31	324779	2722	2.71	1:54:40		mPG+WSA	0/0/5	2.37	330634	20209	7.31	N/A
	QPLACE+WSA	1/0/0	2.11	300843	0	0.53	52:42		QPLACE+WSA	1/0/0	2.21	318476	0	2.61	1:38:56
ibm07	Dragon-fd+WSA	4/1/0	4.24	568607	0.2	0.34	1:34:53	ibm07	Dragon-fd+WSA	2/3/0	4.57	603704	6.4	2.21	4:01:08
-easy	CAPO+WSA	5/0/0	4.46	539314	0	0.39	1:32:33	-hard	CAPO+WSA	0/5/0	5.03	613347	87.8	3.78	11:38:58
	FS+WSA	4/1/0	4.11	548362	0.2	0.19	1:00:03		FS+WSA	4/1/0	4.27	579719	3.6	1.33	2:45:30
	mPG+WSA	1/4/0	5.18	642264	41.4	3.62	6:58:00		mPG+WSA	0/0/5	5.14	643011	26043	7.39	N/A
	QPLACE+WSA	1/0/0	4.55	578504	0	0.29	1:22:02		QPLACE+WSA	1/0/0	5.01	626628	0	2.70	3:41:43
ibm08	Dragon-fd+WSA	5/0/0	4.56	675424	0	0.05	1:04:36	ibm08	Dragon-fd+WSA	2/3/0	4.53	703690	3.6	0.22	1:54:45
-easy	CAPO+WSA	5/0/0	4.89	669344	0	0.21	1:37:19	-hard	CAPO+WSA	1/3/1	5.26	741922	107	1.58	8:41:32
	FS+WSA	2/3/0	5.05	708435	0.8	0.37	1:58:41		FS+WSA	2/3/0	4.94	728869	6	0.71	3:25:28
	mPG+WSA	2/3/0	5.13	738545	7.4	0.61	3:31:36		mPG+WSA	1/3/1	5.08	777426	391	1.67	10:36:57
	QPLACE+WSA	0/1/0	5.25	733071	33	0.48	3:23:22		QPLACE+WSA	1/0/0	5.15	739152	0	0.62	2:52:51
ibm09	Dragon-fd+WSA	4/1/0	3.58	577418	0.2	0.02	0:57:08	ibm09	Dragon-fd+WSA	4/1/0	3.51	586582	0.2	0.03	0:58:38
-easy	CAPO+WSA	5/0/0	3.64	544475	0	0.02	0:52:49	-hard	CAPO+WSA	4/1/0	3.63	557845	0.2	0.04	1:01:01
	FS+WSA	4/1/0	3.63	569627	0.2	0.02	0:50:50		FS+WSA	5/0/0	3.65	582507	0	0.04	0:57:59
	mPG+WSA	4/1/0	3.81	595976	0.2	0.04	1:01:03		mPG+WSA	4/1/0	3.85	613259	0.2	0.13	1:14:52
	QPLACE+WSA	1/0/0	4.02	594290	0	0.03	1:00:55		QPLACE+WSA	0/1/0	3.87	603929	1	0.04	1:02:23
ibm10	Dragon-fd+WSA	3/2/0	7.04	901956	0.4	0.03	1:43:20	ibm10	Dragon-fd+WSA	4/1/0	6.80	910968	8.2	0.07	1:31:27
-easy	CAPO+WSA	3/2/0	7.14	870728	0.4	0.04	1:52:26	-hard	CAPO+WSA	2/3/0	7.07	902358	22.6	0.16	4:18:28
	FS+WSA	4/1/0	7.05	887832	0.2	0.02	1:19:55		FS+WSA	5/0/0	6.98	907629	0	0.07	1:53:17
	mPG+WSA	3/2/0	7.25	925947	1.6	0.10	2:06:01		mPG+WSA	1/4/0	7.48	973529	1.4	0.45	4:49:18
	QPLACE+WSA	0/1/0	7.22	908286	1	0.05	2:22:02		QPLACE+WSA	0/1/0	7.39	940473	1	0.13	2:55:11
ibm11	Dragon-fd+WSA	4/1/0	5.34	752803	0.2	0.05	1:14:31	ibm11	Dragon-fd+WSA	5/0/0	5.27	769995	0	0.17	1:31:27
-easy	CAPO+WSA	4/1/0	5.37	708570	0.2	0.04	1:11:20	-hard	CAPO+WSA	4/1/0	5.40	732870	0.2	0.20	1:38:58
	FS+WSA	5/0/0	5.47	740837	0	0.04	1:06:28		FS+WSA	5/0/0	5.40	753572	0	0.11	1:23:39
	mPG+WSA	4/1/0	5.51	773710	0.2	0.10	1:22:34		mPG+WSA	5/0/0	5.77	813468	0	0.61	2:07:40
	QPLACE+WSA	1/0/0	5.95	779237	0	0.10	1:20:40		QPLACE+WSA	0/1/0	6.05	818762	1	0.43	2:37:54
ibm12	Dragon-fd+WSA	3/2/0	9.98	1092302	0.4	0.13	2:46:13	ibm12	Dragon-fd+WSA	1/4/0	9.83	1115038	3.4	0.30	4:18:31
-easy	CAPO+WSA	2/3/0	10.46	1085250	6.2	0.27	4:28:06	-hard	CAPO+WSA	0/5/0	11.18	1118348	11.6	0.65	6:13:47
	FS+WSA	3/2/0	10.39	1099236	0.4	0.22	2:52:37		FS+WSA	3/2/0	10.37	1155815	30.8	0.99	8:59:24
	mPG+WSA	0/0/5	12.59	1317614	59217	6.25	N/A		mPG+WSA	0/0/5	12.34	1356588	111453	10.81	N/A
	QPLACE+WSA	1/0/0	11.82	1200326	0	0.74	5:48:34		QPLACE+WSA	1/0/0	11.02	1211434	0	1.16	11:10:59
summary	Dragon-fd+WSA	52/26/2	0.944×	0.968×	218	0.80×	1.18×	Except for "vlts", which is the average number of violations per run, results in the summary row are normalized with respect to each original tool without applying WSA. See Table 1 for the detailed results of the original tools.							
	CAPO+WSA	48/27/5	0.932×	0.924×	825	0.33×	0.55×								
	FS+WSA	55/15/0	0.983×	0.953×	1105	0.28×	0.45×								
	mPG+WSA	29/20/31	0.920×	0.948×	14633	0.32×	1.23×								
	QPLACE+WSA	10/6/0	0.989×	1.034×	2.4	1.53×	1.50×								