# Algorithms for Simultaneous Satisfaction of Multiple Constraints and Objective Optimization in a Placement Flow with Application to Congestion Control

Ke Zhong and Shantanu Dutt
Department of Electrical and Computer Engineering
Univ. of Illinois at Chicago
Chicago, IL 60607-7053
{kzhong,dutt}@ece.uic.edu

## ABSTRACT

This paper addresses the problem of tackling multiple constraints simultaneously during a partitioning driven placement (PDP) process, where a larger solution space is available for constraint-satisfying optimization compared to post-placement methods. A general methodology of multi-constraint satisfaction that balances violation correction and primary optimization is presented. A number of techniques are introduced to ensure its convergence and enhance its solution search capability with intermediate relaxation. Application of our approach to congestion control modeled as pin density and external net distribution balance constraints shows it effectively reduces overall congestion by 14.3% and improves chip area by 8.9%, with reasonable running time and only 1.6% increase in wire length. As far as we know, this is the first time an approach to congestion reduction *during* placement optimization produced good congestion improvement with very small wire length increase.

## Categories and Subject Descriptors

J.6 [**Computer-Aided Engineering**]:

## General Terms

Algorithms, Experimentation

## Keywords

Multi-constraint satisfaction, partitioning-driven placement, intermediate relaxation, congestion reduction, connector generation, minimization of objective deterioration

## 1. INTRODUCTION

Placement is an important phase of physical design automation for VLSI circuits.With the maturity of deep submicron (DSM) technology, very complex circuits and systems with 10 to 100 million plus transistors can now be realized on a single chip. For these circuits, routability becomes an important issue that needs to be tackled during the placement phase, or subsequent routing can become difficult and inefficient, for example, [8, 9, 12]. Also, in the DSM regime, a number of debilitating electrical and thermal effects like signal crosstalk, increased temperatures and hot-spots become prominent, and negatively impact chip reliability [13]. These effects can be significantly alleviated by proper circuit placement. Thus in the DSM regime, not only is placement required to optimize *primary objectives* (e.g., timing, power and/or wire area), but it also needs to address *secondary requirements* (e.g., routability, uniform thermal distribution and crosstalk bounding) which can be most effectively modeled as *constraints* on the placement process.

A number of excellent placement algorithms and tools have been developed that tackle only a single objective such as wire area, timing, and congestion [1, 6, 4, 8, 9, 14, 11, 10, 15]. There has been little work in developing a general theory and algorithms for exploring the design space to optimize primary objective(s) and satisfy a few design constraints simultaneously [5]. Various important design constraints are thus currently tackled in the post-placement phases where a much smaller solution space is available than during placement, and where there is not much control over possible deterioration of primary optimization objective(s).

To develop our multi-constraint satisfaction algorithm, we will use the partitioning driven placement (PDP) approach to tackle these problems, due to a number of beneficial features: a) Being a divide-and-conquer approach with low time-complexity, it can practically handle huge DSM designs. b) Being a top-down approach, it is cognizant of the state of the optimization objective(s) and constraints at each partitioning level, which enables balanced placement decisions. c) It is inherently very flexible—the gain functions can be appropriately formulated to optimize the objective of interest, and to model the problem accurately by taking various DSM effects into account.

The outline of the rest of the paper is as follows. Sec. 2 describes promising approaches to multi-constraint satisfaction with a view to minimizing deterioration of the primary objective. Here we also discuss relevant past research in constraint-satisfaction methods. Constraint-based modeling of congestion control follows in Sec. 3. Experimental results are discussed in Sec. 4, and conclusions are in Sec. 5.

## 2. MULTI-CONSTRAINT SATISFACTION

Two core issues regarding multiple constraint satisfaction are: a) Reaching feasible constraint-satisfying solutions from infeasible states in a multi-constraint environment where different constraints can be in conflict with one another; b) Minimizing degradation of the primary optimization objective(s) due to constraint satisfaction.

Unless otherwise stated, we will describe our approaches


*DAC 2002,* June 10-14, 2002, New Orleans, Louisiana, USA.

in the context 2-way recursive PDP for standard-cell circuits (although our approaches are extendible to multi-way partitioning and a number of other design styles). A 2-way recursive PDP flow starts with an input circuit and its assigned rectangular layout area (a single *region*). It then recursively bisects each region (horizontally and/or vertically) and its residing (sub)circuit, until subcircuit sizes in every region become sufficiently small. For a standard-cell circuit, its modules are of equal height, and are normally placed into a number of fixed-height rows. Horizontal partitioning steps thus naturally subdivide regions along row boundaries, and separate them into perfectly aligned groups, each covering a unique set of underlying rows.

To study PDP under multiple constraints, we extensively used the standard cell placement tool SPADE [15]. The SPADE approach features a 'simultaneous level processing' paradigm, facilitating direct, more accurate wire length minimization. It produced very promising results on a set of MCNC benchmark circuits [15].

## 2.1 Constraint Categorization and Modeling

To develop a general constraint-satisfaction method, constraints must be classified and modeled in a normalized fashion. Most PDP constraints can be categorized into *balance* and *non-balance* types. In this paper we focus on the former type, which requires the ratio of (aggregate) constraint metric values on the two subsets of a partition to lie within a given (generally small) range. To formally define a balance-type constraint, let the constraint metric values in two subsets of a partition be $B_0$ and $B_1$, their specified ratio be $r : (1-r)$, tolerances be $t_0$ and $t_1$, and their sum be $B$, respectively. Then the following set of inequalities should be satisfied:

$$B_0 \leq B \cdot (r + t_0)\,, \qquad B_1 \leq B \cdot (1 - r + t_1) \tag{1}$$

where $0 < r < 1$, and $0 < \{t_0, t_1\} < 1$. Further, if we divide both sides of (1) by $B$, we get its *normalized form*, with $b_0$ and $b_1$ representing $B_0/B$ and $B_1/B$, respectively:

$$b_0 \leq r + t_0\,, \qquad b_1 \leq 1 - r + t_1 \tag{2}$$

Two primary balance-type PDP constraints, which have been implicitly included in the standard-cell placement tool, SPADE [15], are *local-size* and *row-size* constraints. These two constraints are for balanced distribution of cells into sub-regions and row-sets, respectively. For local-size constraint, $B$ is the sum of cell sizes within a region, while $B_0$ and $B_1$ are total cell sizes of its two subsets. Similarly, for row-size constraint, $B$ is the total cell size of all regions spanning the same underlying set of rows, and $B_0$ ($B_1$) is the combined cell sizes in subsets 0 (subsets 1) of those regions.

Balance constraints can be further divided into *monotonic* and *non-monotonic* classes. For monotonic constraints, move of a cell from its source subset $s$ to its target subset $t$ only increases constraint metric value in subset $t$, and decreases that in subset $s$. For instance, the local-size and row-size constraints are monotonic. Non-monotonic examples include the *pin-density* constraint for congestion control. Pin density is measured as total number if pins within a region divided by its total cell size. When a cell is moved from subset $s$ to $t$, it is possible to increase the pin density of $s$, while decreasing that of $t$. Therefore, the pin density constraint is non-monotonic. Another categorization of balance constraints is based on the invariant nature of $B$ (e.g., for local-size constraint, sum of cell sizes across both subsets remains unchanged at any cell move, while sum of pin densities is variable).

To address a wide range of constraints in a generic manner, we introduce the *normalized cell constraint weight*, or *c-weight* to model contributions of each cell to the various constraint measures. Each cell $u$ has a vector of c-weights, the $i$th element $w_i(u)$ of which corresponds to a particular constraint $c_i$. A c-weight $w_i(u)$ represents how much $u$'s move can change the current distribution of normalized metric values of constraint $c_i$. Specifically, if cell $u$ is moved from subset $s$ to $t$, $b_t(c_i)$ (denoting the $b_t$ measure for constraint $c_i$) changes by $w_i(u)$, while $b_s(c_i)$ changes by $-w_i(u)$.

For local-size (row-size) constraint of a region, c-weights are computed as individual cell sizes relative to total cell size within the region (row-set). They remain constant throughout the cell move process. For any other constraint $c_i$ whose $B$ is not fixed (e.g., pin density constraint), if cell $u$ is to be moved from subset $s$ to $t$, its $w_i(u)$ is computed using the following set of equations:

$$\Delta(c_i) = b_t(c_i) - b_s(c_i)\,, \qquad \Delta'(c_i) = b'_t(c_i) - b'_s(c_i)\,,$$
$$w_i(u) = (\Delta(c_i) - \Delta'(c_i))/2 \tag{3}$$

where $b'_s(c_i)$ and $b'_t(c_i)$ are normalized metric values of constraint $c_i$ for subset $s$ and $t$, respectively, *after* $u$'s move.

Based on Eqn. 3, c-weights of non-monotonic constraints might take negative values, while those of monotonic balance constraints are always non-negative. Signs of the c-weights can be used in the following way to correct constraint violation. For any constraint $c_i$, we term the subset violating Ineq. (1), if any, as its 'overflowing' subset and the other subset as 'underflowing'. The amount of violation is measured as the difference between the actual constraint metric value and its specified upper bound in the overflowing subset. *The move of cell $u$ can reduce the violation of constraint $c_i$ (if any) if $w_i(u)$ is positive and $u$ is moving out of the overflowing subset, or if $w_i(u)$ is negative and $u$ is moving into the overflowing subset.*

## 2.2 Constraint-Driven Optimization with Intermediate Relaxation

Most placement algorithms perform some type of search on the discrete solution space of the problem for the required solutions. If additional constraints are introduced to the problem, the new solution space normally becomes a fragmented subset of the original one. There are two basic paradigms for tackling extra constraints imposed on the search process, *post-placement* or *simultaneous-with-placement* (*with-placement* for short). The post-placement paradigm has been extensively used in past work for tackling many types of constraints (e.g., [12, 14, 16]). In this paper, we espouse the use of the with-placement paradigm.

With-placement methods can be of two types. The first is the simpler formulation of disallowing a search path to traverse any illegal solution. This type of method, which has been employed in almost all previous partitioning and PDP algorithms, is prone to get stuck at local optima. The second approach is to allow the search to navigate through illegal solutions, with the goal of reaching an end solution that is legal and represents an optimal or near-optimal solution. It means the constraint(s) are relaxed temporarily (*intermediate relaxation* introduced in [3]). This solution method can potentially yield global optimum or near global-optimum solution, since it provides a mechanism to virtually connect the disjoint components of a fragmented solution space.

Intermediate relaxation allows two different patterns of violating moves: a) One-step violation: if a move produces a newly-violating solution, any move following it must at least generate a next solution with progressively lesser amount of violation. b) Multi-step violation: a sequence of moves are allowed that may produce increasing amount of violation, up to certain upper bounds, in the interest of better optimizing the primary objective, and this is followed by a series of moves to completely correct the violations, once the violation upper bounds are reached.

The crux of navigating through illegal portions of solu-

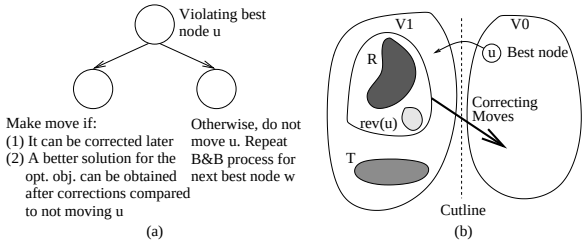


**Figure 1: B&B approach to constraint-satisfaction during partitioning: (a) B&B decision process; (b) The set of correcting and violating moves.**

tion space "profitably" are effective techniques for estimating whether: *(a)* a legal solution can be finally reached, without backtracking, whenever the search, currently at a legal solution, is inclined to visit an illegal one (based on cost improvement); and *(b)* whether the lowest cost legal solution reachable with relaxation (in either single or multiple moves) will have a better cost than the best legal solution reachable without relaxation. The search would then proceed if both (a) and (b) are satisfied. These correspond to the two core issues noted at the beginning of Sec. 2.

## 2.3 Tackling a Single General Constraint

In [3], effective techniques have been developed for formally addressing, for the first time, the problem of partitioning and placement with any **single** given constraint. The challenge in such a problem is to satisfy any given constraint without significantly compromising the optimality of the main objective(s), and [3] introduced the concept of *intermediate relaxation* to achieve this goal. Previous works that studied this problem have had limited success [7].

This promising approach of [3] is a non-backtracking branch-and-bound (B&B) search technique with multi-step violation, which follows the paradigm of making violating moves if both conditions (a) and (b) stated above are satisfied; see Fig. 1(a). In the context of 2-way partitioning with the commonly used and important size balance constraint, an average of 14.5% better cutsizes were obtained with as little as 13% time overhead using these techniques compared to standard methods that disallow violating moves.

Referring to Fig. 1(b), let $V_0, V_1$ be the two subsets into which the current subcircuit is being partitioned (at some level of the PDP hierarchy). Let $T$ be the current set of uncorrected violating moves made into, say, $V_1$, and let $R$ be the set of required correcting moves, say, from $V_1$ to $V_0$ that can correct the violations due to $T$ and will result in the minimum deterioration of the optimization objective. Let $u$ be the current best move to make, say, from $V_0$ to $V_1$. If $u$ does not add to the accumulated violations due to $T$, then we move $u$. Otherwise, we search for the best set of nodes $rev(u)$ (with respect to the optimization objective), $rev(u) \cap R = \emptyset$, that can be moved to compensate for $u$'s violation. We then check if the following inequality holds:

$$gain(u) + gain(R \cup rev(u) \mid u) > gain(R) \qquad (4)$$

where $gain(y)$ is the estimated improvement in the optimization objective due to the move of cell $y$, and for a set $S$ of cells, $gain(S) = \sum_{y \in S} gain(y)$ [1]. Further, $gain(y \mid x)$ is the estimated improvement obtained by moving cell $y$ after $x$ has been moved ($gain(y)$ can be affected by $x$'s move), and $gain(S \mid x) = \sum_{y \in S} gain(y \mid x)$. The LHS (left-hand side) of Ineq. 4 is the estimated objective improvement on pursuing the path of moving $u$ followed by $R \cup rev(u)$ to correct the total violation, while the RHS is the estimated objective improvement obtained by only moving cells in $R$ to correct the current violation; note that a legal partition point must be free of violations. Cell $u$ would then be moved if $rev(u)$ exists and Ineq. 4 is satisfied. Otherwise, the B&B process is repeated for the next best free cell in $V = V_0 \cup V_1$.

[1] The $\sum$ function for computing $gain(S)$ and for use in Ineq. 4, should actually be replaced by the appropriate function $\psi$ that defines the quality of the cutset vis-a-vis the optimization objective. For e.g., $\psi = \sum$ for the min-cut & $\psi$ = weighted max for the timing objectives .

## 2.4 Tackling Multiple Constraints

When we go from single to multiple constraints, the complexity of constraint-satisfying algorithms (with intermediate relaxation) goes up by at least an order of magnitude. This is so, because multiple constraints can be mutually conflicting; i.e., if we correct the violation of one, we could end up worsening that of another. Also, with more constraints, it becomes more difficult to control the compromise in primary objective(s) during violation correction stages. We describe here a number of techniques for dealing with issues of conflicting constraints and with trade-off estimates between gain and constraint correction amount when determining a set of moves to correct multiple constraint violations.

### 2.4.1 Balancing Constraint Violation Correction and Primary Objective Optimization

The SPADE algorithm relies on global gain-based priority queues to select cell moves, and avoids any row-size and local-size violation [15]. Now, with intermediate relaxation of multiple constraints, additional priority queues are introduced for each constraint $c_i$ at its effective level (for row-size, at row level; for local-size, at regional level, etc.). Cells in each such priority queue are sorted according to some mixed measure based on their gains and contributions to violation reduction, making it easier to find cells that have reasonable gains and can help resolve violations (if any) quickly. For any non-monotonic constraint, cells are sorted into two priority queues (with different signs) in each subset, so that no matter which one is overflowing, the algorithm can quickly find correcting cells to move. Also, when violations are present, there is always a *critically violated* constraint $c_m$ that has the max amount of (normalized) violation. While looking for a cell $u$ to reduce $c_m$'s violation, it is also necessary to check $u$'s impact (as indicated by its c-weights) on other constraints lest some other violations grow too much.

Based on the above consideration of balancing optimization objective improvement and violation correction, we introduce the following *gain-c-weight* measure $GW(u, c_i)$ (each constraint-based priority queue is sorted by decreasing $GW(u, c_i)$ measure). If there are $k$ constraints, then for each constraint $c_i$,

$$GW(u, c_i) = (g(u) - g_{th}) \cdot (1 + \frac{\alpha \cdot |w_i(u)|}{max_j |w_j(u)|}) \ \textbf{if } g(u) \geq g_{th}$$
$$GW(u, c_i) = (g(u) - g_{th}) \cdot (1 - \frac{\alpha \cdot |w_i(u)|}{max_j |w_j(u)|}) \ \textbf{if } g(u) < g_{th} \qquad (5)$$

where $g(u)$ is the gain of $u$, $g_{th}$ is a threshold on cell gains, $\alpha$ is a constant for trade-off between gain and c-weight considerations, and $1 \leq \{i, j\} \leq k$. The threshold $g_{th}$ is designed to penalize cells with gains below it. The max operator in the denominator helps convergence in the violation correction process, since a cell with relatively higher $w_i$ (among its other c-weights) is more likely to be chosen if $c_i$ is the current critically violated constraint.

A cell selection algorithm using the above data structures and gain-c-weight measures is depicted in Fig. 2. When violations occur, the algorithm refers to relevant positive and/or negative weight priority queues for selecting the best correcting move. Note that no move is allowed to produce a higher amount of violation than the current critical amount (to be discussed next).

**Algorithm** `Cell_Select`() /* multi-constraint cell selection */
  **if** (current state is feasible) **then** /* any move allowed */
    $Bc$ = the best cell from gain-based priority queues;
  **else** /* at least one constraint is already violated */
    Get the type $c_m$, amount $v_a$, overflowing subset $s_o$ and underflowing subset $s_u$ of the current critical violation;
    Locate relevant positive-weight queue $q^+$ in $s_o$, and negative-weight queue $q^-$ in $s_u$ for $c_m$; /* $q^-$ exists for non-monotonic constraints only */
    $Bc$ = the cell with maximum gain-c-weight value from $q^+$ and/or $q^-$;
    **if** ($Bc$ produces new critical violation $\geq v_a$) **then**
      Pick the next best cell from $q^+$ (or $q^-$) as the new $Bc$, whose post-move critical overflow $< v_a$;
  **return** $Bc$;

**Figure 2: Multi-constraint cell selection algorithm.**

### 2.4.2 Mechanisms to Ensure Fast Convergence to Feasible States

With increasing number of constraints, the problem of convergence becomes critical even under one-step violation. Cell selection based on the mixed gain-c-weight measure cannot ensure correctability of violation (not to mention a speedy correction). Hence, the following rules and techniques are introduced to help solve the convergence problem.

⋆ *Decreasing violation:* Any correcting move should produce a new amount of critical violation strictly less than that of the current violation. Such a rule maintains a trend of convergence, but does not guarantee its speed. This is included as part of the cell selection process in Fig. 2.

⋆ *Feasibility estimation:* Whenever a candidate move introduces a fresh violation, the possibility of correcting the new violation within a certain number of steps is estimated. The estimation process tentatively accepts the violating move, and repeatedly looks for the best cell $v$ to resolve the current most critical violation (in the same way as the cell selection algorithm in Fig. 2). These cells are then 'virtually' moved (they are moved, but no cell gains are updated) to correct violations. Only when violations are completely resolved within a certain number of 'virtual' moves, can the initial violating move be actually accepted. This can help us exclude those moves that can lead to a very long (if not impossible) correction process.

⋆ *Learning-based backtracking:* Sometimes existing violations are not corrected after the number of cell moves reaches a user-specified upper bound. Our backtracking mechanism then divides the current sequence of violating moves into sub-sequences, each focusing on resolving critical violations on a certain type of constraint. It first backtracks to the starting point of the current sub-sequence (focusing on, say, constraint $c_i$) and selects an alternative cell $v$ within $c_i$'s priority queue(s). Cell $v$ is required to produce a low enough violation on $c_i$ so that it can be resolved before the move process hitting the move limit again, assuming the same violation reduction gradient with the failed sub-sequence of moves (this is where learning comes into the picture). Subsequent cells are then picked using the selection algorithm of Sec 3.4.1. If violation still cannot be resolved, the move process backtracks one more level to the starting move of the immediately preceding sub-sequence, and so on. Finally, if backtracking fails for all sub-sequences, the initial violating move is revoked, and replaced by a feasible move (if any).

In summary, the above three mechanisms in combination ensure convergence to feasible PDP results, within a user-specified number of cell moves.

### 2.4.3 Intermediate Relaxation of Multi-Step Violation under Multiple Constraints

All discussions in Sec. 2.4 so far assume a one-step violation approach for multiple constraints, and convergence ensuring mechanisms require that the amount of critical violation decrease for every rectifying move. Multi-step violation based relaxation, on the other hand, allows temporarily increasing violation, and more thorough exploration of the solution space for better results.

Even under multiple constraints, the basic issues for multi-step relaxation are still centered around violation correctability and minimal primary objective deterioration (which requires good gain estimation methods). Also the same terminologies in Sec. 2.3 apply here: $T$ for the set of violating moves that are not corrected, $R$ for the set of required correcting moves due to $T$, and $rev(u)$ for the set of best correcting moves for the current violating move.

For the violation correctability problem, the feasibility estimation scheme for one-step violation, with necessary adaptations, provides a good solution. For any candidate move $u$ that *increases* the current critical violation (otherwise the move is allowed unconditionally), as long as the upper bounds on critical violation are not reached, the algorithm proceeds to determine $rev(u)$. To do that, it repeatedly looks for the best free cell $v$ to rectify the current critical violation, and adds $v$ to $rev(u)$. Any such cell $v$ is required to decrease the current critical violation (for fast convergence). If the final number of cells in $rev(u)$ is within a certain limit, $u$'s violation is deemed correctable.

Once $rev(u)$ is decided, the *only* thing we need to do for gain estimation is to check if that Ineq. 4 holds true. If so, move of $u$ is accepted. Hence, gain estimation is *independent* of the number of constraints involved. Note that in general $rev(u)$ can include cells from both subsets simultaneously.

To develop a high-quality gain estimation scheme, it is very important to estimate the aggregate gain of a set of moves accurately. Two special cases are noteworthy: a) some cells are connected to a common net in one subset; b) some cells share a net in both subsets. In the former case, the combined effect of moving those cells should be considered. For the latter, cell gains from different subsets should (partially) cancel each other out.

## 3. CONSTRAINT-BASED CONGESTION REDUCTION

Some previous work on congestion reduction for placement include [12], [14] and [9]. It is concluded in [12] that inclusion of congestion cost into the primary objective generally does not produce better results (in terms of both wire length and congestion) than a pure wire length objective. Thus, in [12, 14], congestion reduction is performed in a post-processing stage using a number of local improvement methods on wire-length optimized placement. In [9], congestion is estimated after each level of partitioning, and the results are used to influence the quadratic placer at next level. Thus, congestion reduction is also performed in a post-processing manner, albeit level by level.

In our approach, congestion is modeled as a set of constraints on the PDP process so that congestion reduction can be achieved simultaneously with wire length minimization (an *in-processing* approach). The two constraints considered for congestion control are as follows: *(i)* Balance the pin density of each resulting subcircuit of the current partition, as has been explained in Sec. 2.1. This constraint keeps connection density in all regions of the chip evenly distributed, and can thus control overall congestion. *(ii)* Partition a subcircuit so as to distribute its external nets (parallel to the current cutline) among its two subsets in proportion to their dimensions. As shown in Fig. 3, this constraint yields a uniform net distribution across routing channels, and hence a more routable placement. Imposing this constraint is equivalent to performing approximate *global route planning* as we are deciding in a top-down manner through

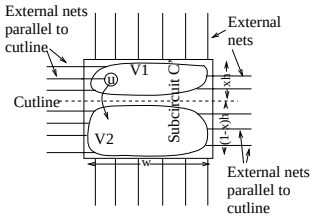


**Figure 3: Proportionate distribution of external nets into subcircuit $C'$ when bi-partitioning $C'$ with a size balance ratio of $x : (1-x)$ for low-congestion global route planning.**

which channels each net should cross to minimize wiring congestion.

The key to modeling external-net constraint is to keep track of the most likely crossing points of external nets in any region, as the PDP flow proceeds to finer levels of partitioning. To this end, a *connector* is introduced to indicate cross-region connections for any pair of net segments (a *net segment* is the part of a net spanning a particular region) that are likely to be directly connected to each other. Specifically, two net segments of a common net in adjacent containing regions are linked to a connector if the likely route for the net will connect these two segments *directly* through a routing channel. The connector is then positioned at the intersection of a rectilinear route connecting its pair of net segments and the shared boundary of containing regions. For example, in Fig. 4(a), two net segments $ns_0$ and $ns_1$ are expected to be connected directly to each other after partitioning, and a connector $x$ is generated at the cutline.

After each level of partitioning, we generate *new* connectors and/or update *existing* ones based on the resultant cell distribution of each net. As seen in Fig. 4(a), a new connector is automatically generated at the cutline whenever a net segment $ns_i$ is locally cut. Existing connectors, on the other hand, have to be re-assigned to new subsets and subsegments at the end of a partitioning process. An existing connector is associated with two net segments, so its new affiliations only depend on the partitioning status of these two segments (in subset 0, subset 1, or cut). Based on this fact, we identify one sub-segment from each associated net segment that are physically (rectilinearly) closest to each other, and re-assign the connector to them. As an example, the only sub-segment of $ns_1$, $ns_{11}$, is closest to the right sub-segment $ns_{01}$ of $ns_0$ in Fig.4(b) as a result of the current level of partitioning (with the vertical dashed cutline), and the existing connector $x$ is re-assigned to them. In some cases there can be two equal-length routes, like in Fig. 4(c) or (d), which lead to two possible new locations. In such cases we will pick a location that gives a better balanced final distribution of external nets. In general, existing connectors are 'inherited' down the PDP hierarchy. Note that in standard-cell placement scenarios, cells have to be abutted horizontally, leading to mis-aligned regions (as shown in Fig. 4(e)). Such *horizontal deviations* may require generation of additional connectors along the dimension *parallel* to the current cutline, if a rectilinear connection between the two closest sub-segments crosses region edges or cutlines other than the one where the existing connector is located.

After connector generation, the next level of partitioning can be started. During the partitioning process, we dynamically determine possible *locations* of each existing connector affiliated to regional edges *perpendicular to* the current cutline, in the same way as we determine its final location after partitioning at this level is completed. If a location on the region boundary of either subset is possible (like in Fig. 4(c)), both are accepted with a 0.5 probability. With locations known, a connector is counted as 1 in its residing subset (those with 0.5 probabilities are counted as 0.5 in both possible subsets). The external-net constraint says that connector counts must be balanced in any region, with the same ratio as local-size balances. This is a balance constraint that can be readily incorporated into our multi-constraint satisfaction framework.

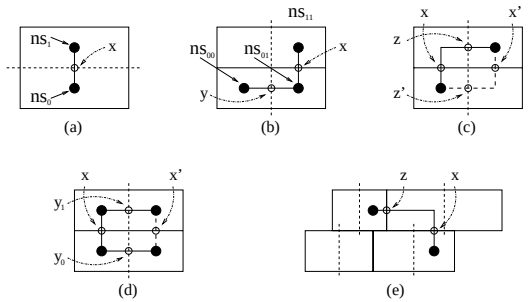


**Figure 4: Examples of connector generation for global route planning. Current cutlines are shown as dashed lines, net segments as dark dots, and connectors as circles. Most likely routes are in solid forms, and possible alternative routes in dashed forms. (a) A new connector $x$ is generated for net segments $ns_0$ and $ns_1$ after partitioning. (b) $x$ is then 'inherited' after partitioning of next level, connected to new sub-segments $ns_{01}$ and $ns_{11}$; connector $y$ is newly generated. (c) In this case, $x$ can be re-assigned to two possible locations ($x$ or $x'$), and an additional connector ($z$ or $z'$) is generated accordingly to complete a rectilinear route. (d) There are again two possible locations for $x$, besides two new connectors $y_0$ and $y_1$. (e) Horizontal deviations may require generation of additional connector(s) $z$ while updating the location of an existing connector $x$.**

# 4. EXPERIMENTAL RESULTS

To test the efficacy of our proposed multi-constraint-satisfaction framework, we used an implementation of the SPADE algorithm based on the SHRINK-PROP [2] partitioner modified for wire length gains. We then developed the following two algorithms, both of which can be incorporated into SPADE: (1) An algorithm that allows one-step violation, and performs feasibility estimation and learning-based backtracking for fast convergence. (2) The second algorithm is an enhanced version of the first, featuring multi-step violation based intermediate relaxation. We conducted experiments first with SPADE alone (implicitly with local-size and row-size constraints), and then the above two multi-constraint (MC) algorithms with the inclusion of the two congestion constraints described in Sec. 3.

Experiments were run on ten MCNC and five IBM-PLACE standard-cell benchmarks whose characteristics are given in Table 1. All experiments were conducted on a 550MHz Pentium-III Linux workstation. Wire length (WL) results are reported in meters, and run times in seconds. The two congestion-related constraints are denoted as *p-dns* (pin-density) and *n-ext* (external-net). An overall congestion metric *ovfl* (overflow) is included to measure total exceeding of wiring capacity (supply) in all regions. Wiring demand at any edge of a region is estimated as the number of connectors along that edge. To determine wiring capacities, we ran SPADE (with local-size and row-size constraints only), performed connector generation, and then collected the average wiring density (number of net crossings at an edge divided by its length) across all regions. We then derived the maximum allowed wiring density as $(1+\epsilon) \cdot log(P)/K$ times the average density, where $\epsilon$ and $K$ are adjustable constants, and $P$ is the total number of pins of a circuit. The $log(P)$ factor stems from our expectation that larger circuits with higher standard deviations in regional wiring density tend to require a higher channel capacity. Wiring

| Circuit | Circuit Statistics | | | PDP w/o p-dns or n-ext | | |
|---|---|---|---|---|---|---|
| | cell | net | row | WL | time | ovfl |
| fract | 125 | 147 | 6 | 0.021 | 1 | 15 |
| prim1 | 752 | 902 | 16 | 0.74 | 28 | 72 |
| struct | 1888 | 1920 | 21 | 0.255 | 68 | 161 |
| prim2 | 2907 | 3029 | 28 | 3.05 | 312 | 228 |
| biomed | 6417 | 5742 | 46 | 1.25 | 549 | 402 |
| in2 | 12142 | 13419 | 72 | 12.01 | 2565 | 596 |
| in3 | 15059 | 21939 | 54 | 36.09 | 4073 | 498 |
| avq_s | 21854 | 22124 | 80 | 5.40 | 3017 | 355 |
| avq_l | 25114 | 25384 | 86 | 5.91 | 3519 | 461 |
| golem3 | 99932 | 144949 | 192 | 19.94 | 24340 | 1023 |
| ibm01 | 12028 | 11507 | 64 | 3.95 | 2730 | 276 |
| ibm02 | 19062 | 18429 | 64 | 11.81 | 6377 | 632 |
| ibm03 | 21879 | 21621 | 64 | 10.50 | 6983 | 384 |
| ibm04 | 26332 | 26163 | 64 | 15.14 | 8081 | 483 |
| ibm05 | 29347 | 28446 | 64 | 37.77 | 12993 | 626 |
| Total | – | – | – | 163.83 | 75636 | 6212 |
| Imprv. | – | – | – | – | – | – |

**Table 1: The left columns show the MCNC and IBM-PLACE benchmark statistics. The right columns are for SPADE results without congestion related constraints.**

capacity of a regional edge is then the product of the edge length and maximum wiring density. In the experiments we used $\epsilon = 0.2$, and $K = 6.14$ (equal to logarithm of the pin number of circuit *fract*), which generally give reasonable total overflow without congestion control. All of our reported results here are obtained when the PDP process ends with no region containing more than 16 cells. A direct comparison of our results with those of [9, 12, 14] is difficult, since their congestion measurements are obtained after routing (global and/or detailed) is done, and there is a lack of general methods to determine wiring supply values for various benchmark circuits.

As shown in Table 2, by using of the first MC algorithm with feasibility estimation and learning-based backtracking mechanisms (with $\alpha = 0.5$), we obtained significant improvement on congestion, with very little WL increase. A separate experiment revealed that individually the pin-density and external-net balance constraints produced 5.6% and 11.2% reduction in congestion respectively.

Overall, we achieved about 14.3% reduction in total overflow, and above 20% improvement for some circuits, at the cost of about 1.6% average increase in wire length. We also obtained 10.5% improvement in the sum of maximum horizontal wiring demand per channel (not shown here due to page limit), which translates to at least $10.5 - 1.6 = 8.9\%$ chip area reduction. These results are quite promising. In comparison, previous methods attempting to reduce congestion during placement have generally obtained worse congestion as well as wire length. To the best of our knowledge, we are the first to show good congestion improvement with very small degradation in WL, by performing with-placement congestion control.

The second algorithm introduced intermediate relaxation with multi-step violation, and resulted in more than 50% less wire length increase compared to algorithm 2 (the deterioration is reduced from 1.6% to 0.7% after relaxation).

# 5. CONCLUSION

This paper has made two major contributions. The first is a general methodology to tackle multiple extra-objective constraints *during* a PDP optimization process. It includes techniques for quickly resolving violation of multiple constraints, and well-balanced consideration of multi-constraint satisfaction and primary objective preservation. The second contribution is a modeling of congestion control using pin-density and external-net balance constraints, and a technique for approximate global route planning (i.e., connector generation). Experiments on congestion reduction using these two constraints demonstrate the efficacy of both our general MC algorithms and constraint-based congestion control method. We achieved 14.3% reduction in congestion, at a wire length cost of only 1.6%, which was further reduced to 0.7% by intermediate relaxation with multi-step violations.

| Circuit | PDP w/ p-dns and n-ext FE+BT | | | PDP w/ p-dns and n-ext FE+BT, relaxation | | |
|---|---|---|---|---|---|---|
| | WL | time | ovfl | WL | time | ovfl |
| fract | 0.020 | 2 | 12 | 0.019 | 2 | 13 |
| prim1 | 0.76 | 33 | 60 | 0.75 | 38 | 62 |
| struct | 0.260 | 86 | 133 | 0.250 | 96 | 139 |
| prim2 | 3.12 | 381 | 200 | 3.11 | 423 | 198 |
| biomed | 1.26 | 656 | 356 | 1.24 | 755 | 334 |
| in2 | 12.26 | 3099 | 484 | 12.11 | 3422 | 503 |
| in3 | 36.71 | 5110 | 414 | 36.50 | 6042 | 428 |
| avq_s | 5.51 | 3886 | 333 | 5.45 | 4514 | 321 |
| avq_l | 6.07 | 4570 | 391 | 5.94 | 5433 | 403 |
| golem3 | 20.18 | 31979 | 884 | 20.05 | 37403 | 910 |
| ibm01 | 3.97 | 3459 | 232 | 3.96 | 3857 | 236 |
| ibm02 | 12.06 | 7998 | 581 | 11.87 | 9214 | 569 |
| ibm03 | 10.69 | 9384 | 332 | 10.63 | 10548 | 346 |
| ibm04 | 15.39 | 10517 | 413 | 15.19 | 12104 | 397 |
| ibm05 | 38.11 | 16795 | 496 | 37.94 | 19813 | 504 |
| Total | 166.41 | 97955 | 5321 | 165.01 | 113664 | 5364 |
| Imprv. | −1.6% | −29.5% | 14.3% | −0.7% | −50.3% | 13.7% |

**Table 2: The left set of results are with pin-density and external-net constraints, as well as feasibility estimation and learning-based backtracking. After introduction of multi-step violation (with relaxation limit set as 1% beyond original tolerances for each constraint), the right set of results show reduced wire length deterioration.**

# 6. REFERENCES


[1] A. E. Dunlop and B. W. Kernighan. A procedure for placement of standard-cell VLSI circuits. *IEEE Trans. CAD*, pp. 92-98, Jan. 1985.

[2] S. Dutt and W. Deng, "Probability-based Approaches to VLSI Circuit Partitioning," *IEEE Trans. CAD*, pp. 534-549, Mar. 2000.

[3] S. Dutt and H. Theny, "Partitioning Around Roadblocks: Tackling Constraints with Intermediate Relaxations", *Proc. ICCAD*, pp. 349-355, 1997.

[4] H. Eisenmann and F. M. Johannes, "Generic Global Placement and Floorplanning," *Proc. DAC*, pp. 269-274, 1998.

[5] H. Esbensen and E. Kuh, "Exploring the design space for building block placements considering area, aspect ratio, path delay and routing congestion", *Proc. Physical Design Workshop*, 1996, pp. 126-133.

[6] D. J-H. Huang and A.B. Kahng, "Partitioning-based standard-cell global placement with an exact objective", *Proc. ISPD*, pp. 18-25, 1997.

[7] R. Kuznar, F. Brglez and K. Kozminski, "Cost minimization of partitions into multiple devices", *Proc. DAC*, pp. 315-320, 1993.

[8] S. Mayrhofer and U. Lauther, "Congestion-driven placement using a new multi-partitioning heuristic", *Proc. ICCAD*, pp. 332-335, 1990.

[9] P.N. Parakh, R.B. Brown and K.A. Sakallah, "Congestion-driven quadratic placement", *Proc. DAC.*, pp. 275-258, 1998.

[10] W-J. Sun and C. Sechen, "Efficient and Effective Placement for Very Large Circuits", *IEEE Trans. CAD*, pp. 349-359, Mar. 1995.

[11] M. Wang, X. Yang and M. Sarrafzadeh, "Dragon2000: Fast Standard-cell Placement for Large Circuits", *Proc. ICCAD*, pp. 260-263, 2000.

[12] M. Wang, X. Yang and M. Sarrafzadeh, "Congestion Minimization During Placement", *IEEE Trans. CAD*, pp. 1140-1148, Oct. 2000.

[13] W. Wolf, *Modern VLSI Design*, 2nd ed., Prentice-Hall, 1998.

[14] X. Yang, R. Kastner and M. Sarrafzadeh, "Congestion Reduction During Placement Based on Integer Programming", *Proc. ICCAD*, pp. 573-576, 2001.

[15] K. Zhong and S. Dutt, "Effective partition-driven placement with simultaneous level processing and global net views", *Proc. ICCAD*, pp. 254-259, 2000.

[16] H. Zhou and D.F. Wong, "Global Routing with Crosstalk Constraints", *Proc. DAC*, pp. 374-377, 1998.