Highly Efficient Gradient Computation for Density-Constrained Analytical Placement

Jason Cong, *Fellow, IEEE*, Guojie Luo, *Student Member, IEEE*, and Eric Radke

***Abstract*—Recent analytical global placers use density constraints to approximate nonoverlap constraints, and these show very successful results. This paper unifies a wide range of density smoothing techniques called global smoothing and presents a highly efficient method for computing the gradient of such smoothed densities used in several well-known analytical placers. This method reduces the complexity of the gradient computation by a factor of n compared with a naïve method, where n is the number of modules. Furthermore, with this efficient gradient computation, it is able to support an efficient nonlinear programming-based placement framework, which supersedes the existing force-directed placement methods. Experiments show that replacing the approximated gradient computation in mPL6 with the exact gradient computation improves wire length by 15% on the IBM-HB+ benchmark and by 3% on average on the modified International Symposium on Physical Design 2005 (ISPD'05) and ISPD'06 placement contest benchmarks with movable macros. The results also show that the augmented Lagrangian method outperforms the quadratic penalty method with the exact gradient computation.**

***Index Terms*—Iterative solvers, overlap constraints, overlap removal, placement.**

I. Introduction

RECENT analytical global placers use density constraints to approximate nonoverlap constraints and show very successful results in both quality and scalability [2], [4], [5]. The differentiability of both the objective and constraint functions is usually required by analytical solvers. However, the density function is normally not smooth; thus, several smoothing techniques have been proposed and implemented to overcome this problem, including the following: 1) the bell-shaped function [6], [10] to replace the rectangle-shaped modules with differential bell-shaped modules; 2) the smoothing operator defined by the Helmholtz equation [2]; 3) the Gaussian smoothing [4] for the density of fixed modules; and 4) the Poisson equation [5] to transform area distribution to some smoothed potential.

Manuscript received May 27, 2008; revised July 22, 2008. Current version published November 19, 2008. This work was supported in part by the National Science Foundation under Awards CCF-0430077 and CCF-0528583. This paper was presented in part at ISPD 2008. This paper was recommended by Associate Editor G.-J. Nam.

J. Cong and G. Luo are with the Department of Computer Science, University of California at Los Angeles, CA 90095 USA (e-mail: cong@cs.ucla.edu; gluo@cs.ucla.edu).

E. Radke is with the Department of Mathematics, University of California at Los Angeles, CA 90095 USA (e-mail: radke@math.ucla.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2008.2006158

In this paper, we consider the smoothing techniques 2), 3), and 4) listed in the previous paragraph, which we call global smoothing techniques because the smoothed density of a single bin is correlated globally with the original density of every bin. Global smoothing techniques were used by the top placers in the International Symposium on Physical Design 2006 (ISPD'06) placement contest [9], and the contest results indicate that these techniques are effective in achieving high-quality solutions. However, until recently, these techniques did not completely conform to the standard nonlinear programming framework. The method in NTUplace [4] did not use Gaussian smoothing for movable modules but only for fixed modules. The method in Kraftwerk [5] used the smoothed potential as the basis for a force-directed method but does not follow a standard nonlinear programming framework. The method in mPL [2] generalized the force-directed method and used a nonlinear programming formulation and solution technique based on the Uzawa algorithm [1] but could only use a simple approximation of the gradient computation for the smoothed density function.

To adopt these global smoothing techniques into a nonlinear programming framework, a fundamental difficulty arises because of the high complexity of gradient computation of the density penalty function. Unlike the bell-shaped function smoothing technique, where the gradient of the density penalty function can be written down explicitly, the global smoothing techniques do not seem to have any simple analytical form and may require a large amount of numerical computation. This difficulty was the motivation for this paper, which has resulted in the following contributions.

1) We observed the common property of the global smoothing techniques 2), 3), and 4), which makes this paper extensible to handling a large class of smoothing techniques.
2) We derived an equivalent expression for the gradient of the density penalty function, which leads to highly efficient numerical computation and reduces the time complexity by a factor of n compared to a naïve computation.
3) We used our efficient density gradient method in a nonlinear programming framework. We consider this to be a contribution because it is the first time that the density-constrained placement problem with global smoothing technique can be solved exactly in the general nonlinear programming framework. Moreover, we found that the resulting placement method supersedes the force-directed placement methods [2], [5].

0278-0070/$25.00 © 2008 IEEE

Authorized licensed use limited to: IEEE Xplore. Downloaded on December 17, 2008 at 14:51 from IEEE Xplore. Restrictions apply.

4) In particular, we applied our gradient computation to the augmented Lagrangian method in a multilevel placement framework and tested this on the IBM-HB+ benchmark [11]. The application leads to a 15% shorter wire length than mPL6 [3]. It also leads to a 3% wire length improvement, on average, in the modified ISPD'05 [8] and ISPD'06 [9] benchmarks with movable macros.

The remainder of this paper is organized as follows. Section II describes the class of smoothing techniques we are concerned with. Section III defines the density penalty function under two kinds of nonlinear programming methods. Section IV derives an equivalent expression for the gradient of density penalty function that leads to highly efficient computation. Section V discusses practical implementation in a global placer. Section VI presents our experimental results. Finally, conclusions and future work are presented in Section VII. In addition, the appendix section provides further analysis on our computation and a better understanding of force-directed methods.

II. Density and Smoothed Density

We begin with a simplified placement problem for wire length minimization under given target density distribution constraints for all bins. These bin density constraints are normally used to replace the nonoverlap constraints. If the bin density constraints are satisfied or almost satisfied, we consider the global placement stage to be completed and leave the remaining work to the detailed placement phase. Thus, the global placement problem is formulated as

$$\begin{aligned} \text{minimize} \quad & WL(\vec{x},\vec{y}) \\ \text{subject to} \quad & D_{ij}(\vec{x},\vec{y}) = I_{ij} \\ & \text{for every} \quad 1 \le i \le M, \quad 1 \le j \le N. \end{aligned} \tag{1}$$

In this problem, there are n modules. The symbols $\vec{x}$ and $\vec{y}$ are the short-hand notations for the n-dimensional vectors (x_k) and (y_k), respectively, where (x_k, y_k) is the placement of the kth module with width w_k and height h_k.

We assume that the placement region is $[0, a] \times [0, b]$, which is a rectangular area with origin at (0, 0), width a, and height b. To measure the placement density, it is divided into $M \times N$ uniform bins B_{ij}, where $1 \le i \le M, 1 \le j \le M$, with bin width $w_B = a/M$ and bin height $h_B = b/N$. $D_{ij}(\vec{x},\vec{y})$ is the average module density in bin B_{ij} determined by the placement $(\vec{x}, \vec{y})$.

The target density constraints $D_{ij}(\vec{x},\vec{y}) = I_{ij}$ require that the density in each bin B_{ij} be equal to the target density I_{ij}. In general, we can support density inequality constraints as well by transforming them to equality constraints using filler cells, as done in [3].

A. Density

To simplify the following analysis, we assume that we have infinite resolution of the bin structure; thus, the set of bin density $\{D_{ij}(\vec{x},\vec{y})\}$ becomes the function of point density $(D(\vec{x},\vec{y}))(u,\nu)$ defined in $[0, a] \times [0, b]$, which can be computed by the summation of the density contribution of each module as $\sum_{k=1}^{n} D_k(x_k, y_k)$, where

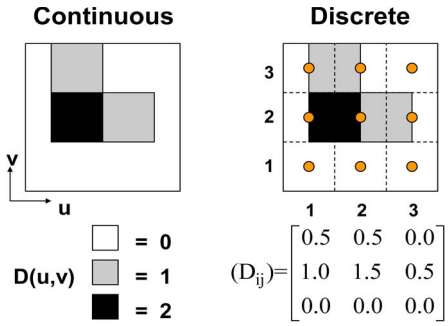

Fig. 1. Continuous and discrete densities.

$$(D_k(x_k,y_k))(u,\nu) = \begin{cases} 1, & u \in [x_k - (w_k/2), x_k + (w_k/2)] \\ & \text{and} \\ & \nu \in [y_k - (h_k/2), y_k + (h_k/2)] \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

A small example with two overlapping modules (with aspect ratio 2 : 1) for the discrete and continuous densities is shown in Fig. 1. The discrete values represent the average bin densities.

The target density $I(u,\nu)$ can also be considered as $\{I_{ij}\}$ with infinite resolution. Thus, we transform the constraints from the discrete bin structures to a continuous density map

$$\begin{aligned} \text{minimize} \quad & WL(\vec{x},\vec{y}) \\ \text{subject to} \quad & (D(\vec{x},\vec{y}))\,(u,\nu) = I(u,\nu) \\ & \text{for every } (u,\nu) \in [0,a] \times [0,b]. \end{aligned} \tag{3}$$

B. Smoothed Density

Note that $(D(\vec{x},\vec{y}))(u,\nu)$ is not differentiable in general. There have been several smoothing techniques (as introduced in Section I, including the Helmholtz equation, the Poisson equation, and Gaussian smoothing; this paper shows that all of these smoothing techniques can be generalized into one class that we shall define in Section IV) as follows, which will be used in the remainder of this paper. Here, we shall first review each existing density smoothing technique.

1) Smoothing by Helmholtz Equation: In [2], the smoothed density $(\widehat{D}_H(\vec{x},\vec{y}))(u,\nu)$ is defined as the solution of Helmholtz equation with zero-derivative boundary conditions

$$\left(\frac{\partial^2}{\partial u^2} + \frac{\partial^2}{\partial \nu^2} - \epsilon\right)\left(\widehat{D}_H(\vec{x},\vec{y})\right)(u,\nu) = -\left(D(\vec{x},\vec{y})\right)(u,\nu). \tag{4}$$

The solution can be written down explicitly through Green's function [13]

$$\left(\widehat{D}_H(\vec{x},\vec{y})\right)(u,\nu) = -\int_0^a\int_0^b (D(\vec{x},\vec{y}))\,(u',\nu')G_H(u,\nu,u',\nu')\,d\nu'du' \tag{5}$$

Authorized licensed use limited to: IEEE Xplore. Downloaded on December 17, 2008 at 14:51 from IEEE Xplore. Restrictions apply.

where

$$G_H(u,\nu,u',\nu') = \frac{1}{ab}\sum_{n=0}^{\infty}\sum_{m=0}^{\infty}\frac{c_n c_m \cos(p_n u)\cos(q_m \nu)\cos(p_n u')\cos(q_m \nu')}{p_n^2+q_m^2+\epsilon} \tag{6}$$

with constants c_n, c_m, p_n, and q_m that only depend on n and m.

2) Smoothing by Poisson Equation: In [5], the smoothed density $(\widehat{D}_P(\vec{x},\vec{y}))(u,\nu)$ is defined as the solution of the Poisson equation with zero derivative at infinity

$$\left(\frac{\partial^2}{\partial u^2}+\frac{\partial^2}{\partial \nu^2}\right)\left(\widehat{D}_P(\vec{x},\vec{y})\right)(u,\nu) = -(D(\vec{x},\vec{y}))(u,\nu). \tag{7}$$

The solution can also be expressed explicitly by Green's function [13]

$$\left(\widehat{D}_P(\vec{x},\vec{y})\right)(u,\nu) = -\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}(D(\vec{x},\vec{y}))(u',\nu')G_P(u,\nu,u',\nu')d\nu' du' \tag{8}$$

where

$$G_P(u,\nu,u',\nu') = \frac{1}{2\pi}\ln\frac{1}{\sqrt{(u-u')^2+(\nu-\nu')^2}}. \tag{9}$$

3) Gaussian Smoothing: The smoothed density $(\widehat{D}_G(\vec{x},\vec{y}))(u,\nu)$ is defined as the convolution between the Gaussian function and the original density $(D(\vec{x},\vec{y}))(u,\nu)$, which is

$$\left(\widehat{D}_G(\vec{x},\vec{y})\right)(u,\nu) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}(D(\vec{x},\vec{y}))(u',\nu')G_G(u,\nu,u',\nu')d\nu' du' \tag{10}$$

where

$$G_G(u,\nu,u',\nu') = \frac{1}{2\pi\sigma^2}e^{-\frac{(u-u')^2+(\nu-\nu')^2}{2\sigma^2}}. \tag{11}$$

4) General Global Smoothing: We observed that the three smoothing techniques described earlier can be generalized into the following density smoothing operation:

$$\left(\widehat{D}(\vec{x},\vec{y})\right)(u,\nu) = \int_0^a\int_0^b(D(\vec{x},\vec{y}))(u',\nu')G(u,\nu,u',\nu')\,d\nu'\,du' \tag{12}$$

where the symmetry property $G(u,\nu,u',\nu') = G(u',\nu',u,\nu)$ is satisfied. Intuitively, $G(u,\nu,u',\nu')$ represents the amount of smoothed density at point (u,ν) created by the original density at point (u',ν'). We call this class of smoothing operation as *global smoothing*, because $G(u,\nu,u',\nu')$ is usually nonzero for every pair of (u,ν) and (u',ν'), which indicates that the influence of the original density to the smoothed density is global.

The discussion and result in the remaining part of this paper will be based on this class of general smoothing operation. Please note that the lower and upper limits of the double integral are bounded in the aforementioned expression. This is only for the convenience of expression, and the following discussion is simple to extend for the unbounded cases.

An important property of the smoothing operator is that two density functions are equal if and only if the corresponding smoothed density functions are equal

$$D_{ij}(\vec{x},\vec{y}) = I_{ij} \Leftrightarrow \widehat{D}_{ij}(\vec{x},\vec{y}) = \widehat{I}_{ij} \tag{13}$$

$$(D(\vec{x},\vec{y}))(u,\nu) = I(u,\nu) \Leftrightarrow \left(\widehat{D}(\vec{x},\vec{y})\right)(u,\nu) = \widehat{I}\ (u,\nu). \tag{14}$$

Using such smoothing operation, we can transform the density constraints to the smoothed density constraints as follows to obtain differentiability.

Discrete smoothed density constraints

$$\begin{aligned}&\text{minimize} && WL(\vec{x},\vec{y})\\ &\text{subject to} && \widehat{D}_{ij}(\vec{x},\vec{y}) = \widehat{I}_{ij}\\ &&& \text{for every } 1\le i\le M,\quad 1\le j\le N.\end{aligned} \tag{15}$$

Continuous smoothed density constraints

$$\begin{aligned}&\text{minimize} && WL(\vec{x},\vec{y})\\ &\text{subject to} && \left(\widehat{D}(\vec{x},\vec{y})\right)(u,\nu) = \widehat{I}\ (u,\nu)\\ &&& \text{for every} \quad (u,\nu)\in[0,a]\times[0,b].\end{aligned} \tag{16}$$

The arc $\widehat{\ }$ is added on top of a density function to differentiate that it is a smoothed density function.

III. Density Penalty Function

The problems specified in (15) and (16) are constrained nonlinear programming problems. Their solution typically involves solving a sequence of unconstrained problems. Two commonly used methods are the quadratic penalty and augmented Lagrangian methods, where the unconstrained problems optimize a combination of wire length and penalty functions on the density constraints. The details will be described in the following sections.

A. Quadratic Penalty Method

The quadratic penalty function for problem (15) is

$$WL(\vec{x},\vec{y}) + \frac{\mu}{2}\sum_{i=1}^{M}\sum_{j=1}^{N}\left(\widehat{D}_{ij}(\vec{x},\vec{y}) - \widehat{I}_{ij}\right)^2 w_B h_B \tag{17}$$

where μ is the quadratic penalty parameter.

Authorized licensed use limited to: IEEE Xplore. Downloaded on December 17, 2008 at 14:51 from IEEE Xplore. Restrictions apply.

Moreover, the quadratic penalty function for problem (16) is

$$Q(\vec{x},\vec{y};\mu) = WL(\vec{x},\vec{y}) + \frac{\mu}{2}\int_0^a\int_0^b \left(\left(\widehat{D}(\vec{x},\vec{y})\right)(u,\nu) - \widehat{I}(u,\nu)\right)^2 d\nu du. \tag{18}$$

This function can be viewed as the limit of the discrete case when the resolution of placement bins becomes infinite. However, more generally, it can be considered in vector space notation, so that problem (16) can be rewritten as

$$\begin{aligned} &\text{minimize} && WL(\vec{x},\vec{y}) \\ &\text{subject to} && \left(\widehat{D}(\vec{x},\vec{y})\right)(*,*) = \widehat{I}\ (*,*) \end{aligned} \tag{19}$$

where $\widehat{D}(\vec{x},\vec{y})(*,*)$ and $\widehat{I}\ (*,*)$ are considered vectors in the vector space $L^2([0,a]\times[0,b])$, which consists of all the 2-D functions, g defined in $[0,a]\times[0,b]$ such that $\int_0^a\int_0^b g^2 d\nu du < \infty$. In this vector space, for any two functions $g_1(*,*), g_2(*,*) \in L^2([0,a]\times[0,b])$, we can define their inner product as follows:

$$g_1(*,*)\odot g_2(*,*) = \int_0^a\int_0^b g_1(u,\nu)g_2(u,\nu)d\nu du. \tag{20}$$

The norm of $g(*,*)\in L^2([0,a]\times[0,b])$ is defined as $\|g(*,*)\| = \sqrt{g(*,*)\odot g(*,*)}$. With these definitions, basic concepts, like the limit and the convergence, can also be defined, and the convergence of the quadratic penalty method in this vector space can be proved seamlessly [7].

Under the notion of vector space, the quadratic penalty function can be written as

$$Q(\vec{x},\vec{y};\mu) = WL(\vec{x},\vec{y}) + P_Q(\vec{x},\vec{y};\mu) \tag{21}$$

where

$$P_Q(\vec{x},\vec{y};\mu) = \frac{\mu}{2}\left\|\widehat{D}(\vec{x},\vec{y}) - \widehat{I}\right\|^2. \tag{22}$$

and their gradients can be computed as follows:

$$\nabla Q(\vec{x},\vec{y};\mu) = \nabla WL(\vec{x},\vec{y}) + \nabla P_Q(\vec{x},\vec{y};\mu) \tag{23}$$

$$\nabla P_Q(\vec{x},\vec{y};\mu) = \mu\left(\widehat{D}(\vec{x},\vec{y}) - \widehat{I}\right)\odot\nabla\widehat{D}(\vec{x},\vec{y}) \tag{24}$$

with scalar μ as the quadratic penalty parameter.

We call $P_Q(\vec{x},\vec{y};\mu)$ the *density penalty function* for the quadratic penalty method and $\nabla P_Q(\vec{x},\vec{y};\mu)$ the *gradient of the density penalty function*.

The algorithmic framework of the quadratic penalty method [12] is given in Fig. 2.

B. Augmented Lagrangian Method

Similar to the quadratic penalty method in vector space, the augmented Lagrangian function for problem (16) can be written as

Given
  $\mu^{(0)} > 0$, tolerance $\tau^{(0)} > 0$,
  and a starting point $(\vec{x}^{(0)}, \vec{y}^{(0)})$;
for $k = 0, 1, 2, \ldots$
  Starting at $(\vec{x}^{(k)}, \vec{y}^{(k)})$,
  Find an approximate minimizer $(\vec{x}^{(k+1)}, \vec{y}^{(k+1)})$,
    such that $\left\|\nabla Q(\vec{x}^{(k+1)}, \vec{y}^{(k+1)}; \mu^{(k)})\right\| \le \tau^{(k)}$;
  if a convergence test is satisfied
    Stop with approximate solution $(\vec{x}^{(k+1)}, \vec{y}^{(k+1)})$;
  end if
  Choose new penalty parameter $\mu^{(k+1)} > \mu^{(k)}$;
  Select tolerance $\tau^{(k+1)}$;
end for

Fig. 2. Algorithm 1—quadratic penalty method.

Given
  $\mu^{(0)} > 0$, tolerance $\tau^{(0)} > 0$,
  a starting point $(\vec{x}^{(0)}, \vec{y}^{(0)})$ and $\lambda^{(0)}$;
for $k = 0, 1, 2, \ldots$
  Starting at $(\vec{x}^{(k)}, \vec{y}^{(k)})$,
  Find an approximate minimizer $(\vec{x}^{(k+1)}, \vec{y}^{(k+1)})$
    such that $\left\|\nabla L_A(\vec{x}^{(k+1)}, \vec{y}^{(k+1)}, \lambda^{(k)}; \mu^{(k)})\right\| \le \tau^{(k)}$;
  if a convergence test is satisfied
    Stop with approximate solution $(\vec{x}^{(k+1)}, \vec{y}^{(k+1)})$;
  end if
  Update $\lambda^{(k+1)} \leftarrow \lambda^{(k)} + \mu^{(k)}(\widehat{D}(\vec{x}^{(k+1)}, \vec{y}^{(k+1)}) - \widehat{I})$;
  Choose new penalty parameter $\mu^{(k+1)} \ge \mu^{(k)}$;
  Select tolerance $\tau^{(k+1)}$;
end for

Fig. 3. Algorithm 2—augmented Lagrangian method.

$$L_A(\vec{x},\vec{y},\lambda;\mu) = WL(\vec{x},\vec{y}) + P_A(\vec{x},\vec{y},\lambda;\mu) \tag{25}$$

where

$$P_A(\vec{x},\vec{y},\lambda;\mu) = \lambda\odot\left(\widehat{D}(\vec{x},\vec{y}) - \widehat{I}\right) + P_Q(\vec{x},\vec{y};\mu) \tag{26}$$

and its gradient is

$$\nabla L_A(\vec{x},\vec{y},\lambda;\mu) = \nabla WL(\vec{x},\vec{y}) + \nabla P_A(\vec{x},\vec{y},\lambda;\mu) \tag{27}$$

where

$$\begin{aligned}\nabla P_A(\vec{x},\vec{y},\lambda;\mu) &= \lambda\odot\nabla\widehat{D}(\vec{x},\vec{y}) + \nabla P_Q(\vec{x},\vec{y};\mu) \\ &= \left(\lambda + \mu\left(\widehat{D}(\vec{x},\vec{y}) - \widehat{I}\right)\right)\odot\nabla\widehat{D}(\vec{x},\vec{y})\end{aligned} \tag{28}$$

with scalar μ as the quadratic penalty parameter and $\lambda(*,*)\in L^2([0,a]\times[0,b])$ as the Lagrangian multiplier.

Similarly, we call $P_A(\vec{x},\vec{y},\lambda;\mu)$ the *density penalty function* for the augmented Lagrangian method and $\nabla P_A(\vec{x},\vec{y},\lambda;\mu)$ the *gradient of the density penalty function*. The algorithmic framework of the augmented Lagrangian method [12] is shown in Fig. 3.

Authorized licensed use limited to: IEEE Xplore. Downloaded on December 17, 2008 at 14:51 from IEEE Xplore. Restrictions apply.

IV. Gradient Computation

The gradient of the density penalty function of either the quadratic penalty method in (24) or the augmented Lagrangian method in (28) has the common form $g \odot \nabla \widehat{D}(\vec{x},\vec{y})$, where the function g equals $\mu(\widehat{D}(\vec{x},\vec{y}) - \widehat{I})$ in the quadratic penalty method and equals $\lambda + \mu(\widehat{D}(\vec{x},\vec{y}) - \widehat{I})$ in the augmented Lagrangian method. It is a $2n$-dimensional vector, whose components consist of $g \odot \partial\widehat{D}(\vec{x},\vec{y})/\partial x_k$ and $g \odot \partial\widehat{D}(\vec{x},\vec{y})/\partial y_k$ for $k = 1, 2, \ldots, n$.

The computation of this gradient is required to be performed hundreds or thousands of times in solving the constrained nonlinear programming problems. However, the computation is not trivial. We will explain why a naïve method is not practical and how we efficiently solve this problem. In the following sections, we assume that the run time for the smoothing operation is $T(n)$ and it is superlinear [of higher order than $O(n)$], as we need to consider every cell during the smoothing operation.

A. Naive Computation

To compute $g \odot \nabla \widehat{D}(\vec{x},\vec{y})$, a naïve method has to compute the components $g \odot \partial\widehat{D}(\vec{x},\vec{y})/\partial x_k$ and $g \odot \partial\widehat{D}(\vec{x},\vec{y})/\partial y_k$ one by one and then expand the inner product for each component

$$g \odot \frac{\partial \widehat{D}(\vec{x},\vec{y})}{\partial x_k} = \int_0^a \int_0^b g(u,\nu) \frac{\partial\left(\widehat{D}(\vec{x},\vec{y})\right)(u,\nu)}{\partial x_k} d\nu du \tag{29}$$

$$g \odot \frac{\partial \widehat{D}(\vec{x},\vec{y})}{\partial y_k} = \int_0^a \int_0^b g(u,\nu) \frac{\partial\left(\widehat{D}(\vec{x},\vec{y})\right)(u,\nu)}{\partial y_k} d\nu du. \tag{30}$$

By discretization, the double integrals are computed through double summation, and the partial derivatives are computed through a certain finite difference scheme.

The detailed description is shown in Fig. 4. In the loop between lines 04 to 11, there are two time-consuming parts. Line 07 consumes $T(n)$ time in each loop, and the computation of $\{\widehat{D}'_{ij}\}$ cannot be reused. Lines 06 and 08 consume $O(MN) = O(n)$ time, assuming the bin number is of the same magnitude as the number of modules. Therefore, the time complexity is $O(n)(T(n) + O(n)) = O(n)T(n)$ for this naïve computation.

B. Efficient Computation

To avoid the time-consuming part of the naïve computation, the equivalent expressions of the inner products $g \odot \partial\widehat{D}(\vec{x},\vec{y})/\partial x_k$ and $g \odot \partial\widehat{D}(\vec{x},\vec{y})/\partial y_k$ are derived for efficient computation and are given in the following theorem.

Theorem 1: Assume that the function $G(u,\nu,u',\nu')$ for the smoothing operator is locally square integrable (e.g., $G \in L^2_{\text{loc}}(R^4)$) and contains a finite number of singularities in any bounded region. Let $D(\vec{x},\vec{y})$ and $\widehat{D}(\vec{x},\vec{y})$ be the density and smoothed density defined in Section II, respectively; then, for any function $g \in L^2([0,a] \times [0,b])$, we have

$$g \odot \frac{\partial \widehat{D}(\vec{x},\vec{y})}{\partial x_k} = \int_{y_k - h_k/2}^{y_k + h_k/2} \left(\widehat{g}\left(x_k + \frac{w_k}{2}, \nu\right) - \widehat{g}\left(x_k - \frac{w_k}{2}, \nu\right)\right) d\nu \tag{31}$$

$$g \odot \frac{\partial \widehat{D}(\vec{x},\vec{y})}{\partial y_k} = \int_{x_k - w_k/2}^{x_k + w_k/2} \left(\widehat{g}\left(u, y_k + \frac{h_k}{2}\right) - \widehat{g}\left(u, y_k - \frac{h_k}{2}\right)\right) du. \tag{32}$$

Proof: Since the smoothing operator is linear, $\widehat{D}(\vec{x},\vec{y})$ can be decomposed to $\sum_{k=1}^{n} \widehat{D}_k(x_k, y_k)$. Thus

$$\frac{\partial \widehat{D}(\vec{x},\vec{y})}{\partial x_k} = \frac{\partial \widehat{D}_k(x_k,y_k)}{\partial x_k} \tag{33}$$

$$= \frac{\partial}{\partial x_k} \int_0^a \int_0^b \left(D_k(x_k,y_k)\right)(u',\nu') G(u,\nu,u',\nu') d\nu' du' \tag{34}$$

$$= \frac{\partial}{\partial x_k} \int_{x_k - w_k/2}^{x_k + w_k/2} \int_{y_k - h_k/2}^{y_k + h_k/2} G(u,\nu,u',\nu') d\nu' du' \tag{35}$$

$$= \int_{y_k - h_k/2}^{y_k + h_k/2} G(u,\nu,x_k + w_k/2, \nu') d\nu' - \int_{y_k - h_k/2}^{y_k + h_k/2} G(u,\nu,x_k - w_k/2, \nu')\, d\nu'. \tag{36}$$

Input: $(\vec{x},\vec{y})$, $\{g_{ij}\}$
Output: $g \odot \nabla\widehat{D}(\vec{x},\vec{y})$
Algorithm:
01: $\{D_{ij}\} \leftarrow$ compute_density $(\vec{x},\vec{y})$;
02: $\{\widehat{D}_{ij}\} \leftarrow$ smooth $(\{D_{ij}\})$;
03: Select small Δx , Δy ;
04: **for** $k = 1, 2, 3, \ldots, n$
05: $x_k \leftarrow x_k + \Delta x$;
06: $\{D'_{ij}\} \leftarrow$ compute_density $(\vec{x},\vec{y})$;
07: $\{\widehat{D}'_{ij}\} \leftarrow$ smooth $(\{D'_{ij}\})$;
08: $g \odot \dfrac{\partial\widehat{D}(\vec{x},\vec{y})}{\partial x_k} \leftarrow \sum_{i=1}^{M}\sum_{j=1}^{N} g_{ij} \dfrac{\widehat{D}'_{ij} - \widehat{D}_{ij}}{\Delta x} w_B h_B$;
09: $x_k \leftarrow x_k - \Delta x$;
10: Compute $g \odot \dfrac{\partial\widehat{D}(\vec{x},\vec{y})}{\partial y_k}$ as in lines 05-09;
11: **end for**

Fig. 4. Algorithm 3—naïve gradient computation for the density penalty function.

Authorized licensed use limited to: IEEE Xplore. Downloaded on December 17, 2008 at 14:51 from IEEE Xplore. Restrictions apply.

Step (33) only keeps the smoothed density of cell k, since the partial derivative of other smoothed density with respect to x_k is zero. Step (34) expands $\widehat{D}_k(x_k, y_k)$ that is defined in Section II-B4. Step (35) drops the integral region where $D_k(x_k, y_k)$ is zero. Step (36) is derived according to the Leibniz integral rule. For a function $G(u, \nu, u', \nu')$ with singularities, the Leibniz integral rule may not apply. See Appendix III for the justification of step (36) in this case. It should be noted that the resulting integrals in step (36) need not exist for every possible (u, ν) pair in the integrated region, but do for almost every (u, ν) pair, and the validity of the following steps shall be explained below. Therefore

$$g \odot \partial \widehat{D}(\vec{x}, \vec{y})/\partial x_k$$

$$= \int_0^a \int_0^b g(u,\nu) \frac{\partial \widehat{D}(\vec{x}, \vec{y})}{\partial x_k} d\nu du \tag{37}$$

$$= \int_0^a \int_0^b g(u,\nu) \left(\int_{y_k - h_k/2}^{y_k + h_k/2} G(u, \nu, x_k + w_k/2, \nu') d\nu' - \int_{y_k - h_k/2}^{y_k + h_k/2} G(u, \nu, x_k - w_k/2, \nu') d\nu' \right) d\nu du \tag{38}$$

$$= \int_{y_k - h_k/2}^{y_k + h_k/2} \left(\int_0^a \int_0^b g(u,\nu) G(u, \nu, x_k + w_k/2, \nu') d\nu du - \int_0^a \int_0^b g(u,\nu) G(u, \nu, x_k - w_k/2, \nu') d\nu du \right) d\nu' \tag{39}$$

$$= \int_{y_k - h_k/2}^{y_k + h_k/2} \left(\int_0^a \int_0^b g(u,\nu) G(x_k + w_k/2, \nu', u, \nu) d\nu du - \int_0^a \int_0^b g(u,\nu) G(x_k - w_k/2, \nu', u, \nu) d\nu du \right) d\nu' \tag{40}$$

$$= \int_{y_k - h_k/2}^{y_k + h_k/2} \left(\widehat{g}\ (x_k + w_k/2, \nu') - \widehat{g}\ (x_k - w_k/2, \nu') \right) d\nu'. \tag{41}$$

Step (37) expands the inner product defined in Section III-A. Step (38) substitutes $\partial \widehat{D}(\vec{x}, \vec{y})/\partial x_k$ by the previous computation. Step (39) changes the order of integrals. Step (40) exchanges the variables of function G because of its symmetric property defined in Section II-B4. Step (41) applies the definition of smoothed density, as in Section II-B4.

Input: $(\vec{x}, \vec{y})$, $\{g_{ij}\}$
Output: $g \odot \nabla \widehat{D}(\vec{x}, \vec{y})$
Algorithm:
01: $\{D_{ij}\} \leftarrow$ compute_density $(\vec{x}, \vec{y})$;
02: $\{\widehat{D}_{ij}\} \leftarrow$ smooth $(\{D_{ij}\})$;
03: $\{\widehat{g}_{ij}\} \leftarrow$ smooth $(\{g_{ij}\})$;
04: **for** $k = 1, 2, 3, ..., n$
05: $x_L \leftarrow x_k - w_k/2$, $x_R \leftarrow x_k + w_k/2$;
06: $y_B \leftarrow y_k - h_k/2$, $y_T \leftarrow y_k + h_k/2$;
07: $g \odot \partial \widehat{D}(\vec{x}, \vec{y})/\partial x_k \leftarrow$ integrate$(\{\widehat{g}_{ij}\}, (x_R, y_B), (x_R, y_T))$ $-$integrate$(\{\widehat{g}_{ij}\}, (x_L, y_B), (x_L, y_T))$;
08: $g \odot \partial \widehat{D}(\vec{x}, \vec{y})/\partial y_k \leftarrow$ integrate$(\{\widehat{g}_{ij}\}, (x_L, y_T), (x_R, y_T))$ $-$integrate$(\{\widehat{g}_{ij}\}, (x_L, y_B), (x_R, y_B))$;
09: **end for**

Function integrate $(\{g_{ij}\}, (x_1, y_1), (x_2, y_2))$
01: Select the number M of intervals dividing the segment $(x_1, y_1) - (x_2, y_2)$;
02: $\Delta x \leftarrow (x_2 - x_1)/M$, $\Delta y \leftarrow (y_2 - y_1)/M$;
03: $I \leftarrow$ interpolate $(\{g_{ij}\}, (x_1, y_1))/2$ + interpolate $(\{g_{ij}\}, (x_2, y_2))/2$;
04: **for** $k = 1, 2, 3, ..., M - 1$
05: $x \leftarrow x_1 + k\Delta x$, $y \leftarrow y_1 + k\Delta y$;
06: $I \leftarrow I +$ interpolate $(\{g_{ij}\}, (x, y))$;
07: **end for**
08: $I \leftarrow I \cdot \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}/M$;
09: **return** I .

Function interpolate $(\{g_{ij}\}, (x, y))$
assume $(a_i, b_j) - (a_{i+1}, b_{j+1})$ is the bin containing (x, y) ;
return $g_{i,j}(a_{i+1} - x)(b_{j+1} - y)/(w_B h_B)$
$+ g_{i,j+1}(a_{i+1} - x)(y - b_j)/(w_B h_B)$
$+ g_{i+1,j}(x - a_i)(b_{j+1} - y)/(w_B h_B)$
$+ g_{i+1,j+1}(x - a_i)(y - b_j)/(w_B h_B)$.

Fig. 5. Algorithm 4—efficient gradient computation for the density penalty function.

The changing order of integration at step (39) is valid for G containing singularities. Since g and G are in L^2 in the integrated region, it follows by Hölder's inequality [15] that $gG \in L^1$ in the integrated region. Thus, we may invoke the Fubini–Tonelli theorem [15] to justify the change of order of integration in step (39). It also follows from the Fubini–Tonelli theorem that the resulting iterated integral is valid, despite the integrals in step (36) possibly not existing for every (u, ν) pair. Please refer to Appendix II for the descriptions of Hölder's inequality and Fubini–Tonelli theorem.

In the same way, we can also derive (32) for $g \odot \partial \widehat{D}(\vec{x}, \vec{y})/\partial y_k$. ■

The key insight of (31) [or (32)] is that the integral of the product $g(u, \nu) \odot (\partial \widehat{D}(\vec{x}, \vec{y})/\partial x_k)(u, \nu)$ over the placement region is equivalent to the difference of the integrals of $\widehat{g}\ (u, \nu)$ along a pair of opposing edges on the module boundary. The naïve computation has to compute $\partial \widehat{D}(\vec{x}, \vec{y})/\partial x_k$ for each k, but with (31), we only need to compute $\widehat{g}\ (u, \nu)$ once and reuse it for all the modules. Thus, we give a highly efficient computation of $g \odot \nabla \widehat{D}(\vec{x}, \vec{y})$ in Fig. 5. The function interpolate

Authorized licensed use limited to: IEEE Xplore. Downloaded on December 17, 2008 at 14:51 from IEEE Xplore. Restrictions apply.

$(\{g_{ij}\},(x,y))$ computes $g(x,y)$ at any (x,y) by bilinear interpolation. Moreover, the numerical integration is computed by the function integrate$(\{g_{ij}\},(x_1,y_1),(x_2,y_2))$ for (31) and (32).

During implementation, $g(x,y)$ is represented as a matrix instead of a continuous function, so that the function value at (x,y) is computed by bilinear interpolation from the neighboring points stored in the matrix.

Theorem 2: The computational complexity in Fig. 5 is $T(n)$, which is no greater than the complexity of the smoothing operation.

Proof: Lines 01 to 03 consume $T(n)$ time for smoothing. Integrals are computed numerically in each loop between line 04 and line 09. Because the lower and upper limits of these integrals depend on the module size, which is a constant in the problem, they only require β time to compute each integral and $O(n)$ time in total. Therefore, $T(n)+O(n)=T(n)$ is the time complexity for Fig. 5. ■

The advantage of Fig. 5 is that $\{\widehat{g}_{ij}\}$ can be reused in every loop, so that we only need to smooth $\{g_{ij}\}$ once. This algorithm reduces the run time by a factor of n in terms of computational complexity compared to the naïve method.

V. Practical Implementation in a Global Placer

Integrating the gradient computation method in a global placer involves the practical implementation of the quadratic penalty method or the augmented Lagrangian method. The following sections will give detailed considerations for implementing Figs. 2 and 3.

A. Density Grids and Smoothed Density

The placement region is scaled into a unit square, and the density function is implemented with discrete bin grids. The grids are constructed in such a way that the grid size is equal to the average area of "most" cells, which consist of 90% of the movable macros and movable standard cells and excludes the largest 5% and the smallest 5%.

The smoothing operator is selected to be the square root of the Helmholtz smoothing operator. According to Section IV-B, a twice-smoothed density has to be computed; thus, the Helmholtz smoothing is performed on the original density. The parameter ε referred in Section II-B-1 is set to be 100.

B. Step Size Selection

In each iteration, the solver searches for a minimizer of the nonlinear function $F(\vec{x},\vec{y})=WL(\vec{x},\vec{y})+P(\vec{x},\vec{y})$ for both methods, where the penalizing term $P(\vec{x},\vec{y})$ represents $P_Q(\vec{x},\vec{y};\mu)$ in the quadratic penalty method and represents $P_A(\vec{x},\vec{y},\lambda;\mu)$ in the augmented Lagrangian method. Given an initial solution $(\vec{x}^{(k)},\vec{y}^{(k)})$ from the last iteration, the gradient descent method can be used to minimize $F(\vec{x},\vec{y})$ by iteratively updating

$$(\vec{x}_{\text{new}},\vec{y}_{\text{new}}) \leftarrow (\vec{x}_{\text{old}},\vec{y}_{\text{old}}) - \alpha\nabla F(\vec{x}_{\text{old}},\vec{y}_{\text{old}}) \tag{42}$$

until it converges to a stationary point $(\vec{x}^{(k+1)},\vec{y}^{(k+1)})$, which is the local minimizer of the function $F(\vec{x},\vec{y})$.

In order to reduce the run time, a constant step size is used instead of line search. If the gradient descent method cannot converge with step size α for the initial solution $(\vec{x}^{(k)},\vec{y}^{(k)})$, a smaller step size (e.g., 0.6α) will be tried.

In general, the constant step size α at the kth iteration is

$$\alpha = k\eta^m\alpha_0 \tag{43}$$

where α_0 is the initial step size, m is the number of disconvergent trials from the first iteration to the kth iteration, and $\eta<1$ is the factor to reduce the step size. The current iteration number k is multiplied as a heuristic to provide opportunity to increase the step size again. During the implementation, $\alpha_0=1$ and $\eta=0.6$ are used.

C. Penalty Factor Update Scheme

This update scheme of the penalty factor μ is applied to both the quadratic penalty and augmented Lagrangian methods. The basic idea is to increase μ when the overlap removal becomes slow

$$\mu^{(k+1)} \leftarrow \begin{cases} \gamma\mu^{(k)}, & \text{overlap removal is slow} \\ \mu^{(k)}, & \text{otherwise} \end{cases} \tag{44}$$

where $\gamma>0$ and $\gamma=1.2$ are used during implementation.

To facilitate the decision whether overlap removal is slow, two concepts are used: The percentage of cell overlap OVL is defined as

$$\text{OVL} = \frac{\int_0^1\int_0^1 \max\left(0, D(u,\nu)-I(u,\nu)\right)\, d\nu du}{\int_0^1\int_0^1 D(u,\nu)\, d\nu du} \tag{45}$$

and the overlap reduction rate r is defined as

$$r = (\text{OVL}_{k-1}-\text{OVL}_k)/\text{OVL}_{k-1}. \tag{46}$$

We consider that the overlap removal is slow, when (OVL $>25\%$ and $r<5\%$) or (OVL $\le 25\%$ and $r<0.5\%$).

D. Lagrangian Multiplier Update Scheme

Different from the scheme in Fig. 3, the Lagrangian multiplier is not updated every iteration. Instead, it updates only when the penalty factor μ is not increasing, which means that the Lagrangian multiplier is updated only when the overlap removal is fast enough.

To make the iterative method more stable, a damping factor β is introduced in the update

$$\lambda^{(k+1)} \leftarrow \lambda^{(k)} + \beta\mu^{(k)}\left(\widehat{D}\left(\vec{x}^{(k+1)},\vec{y}^{(k+1)}\right)-\widehat{I}\right). \tag{47}$$

During implementation, $\beta=\gamma-1$ is used, where γ is defined in Section V-C.

Authorized licensed use limited to: IEEE Xplore. Downloaded on December 17, 2008 at 14:51 from IEEE Xplore. Restrictions apply.

TABLE I
BENCHMARK STATISTICS

IBM-HB+				modified ISPD'05 & ISPD'06			
circuit	#cell	#net	WS%	circuit	#cell	#net	WS%
ibm01	911	5829	20.1	adaptec2	254616	266009	21.4
ibm02	1471	8508	20.0	adaptec4	496141	515951	37.3
ibm03	1289	10279	21.2	bigblue1	277636	284479	45.8
ibm04	1584	12456	20.0	bigblue2	557962	577235	38.1
ibm06	749	9963	20.0	bigblue3	1096908	1123170	14.3
ibm07	1120	15047	20.0	bigblue4	2177449	2229886	34.7
ibm08	1269	16075	20.3	adaptec5	843224	867798	21.3
ibm09	1113	18913	20.1	newblue1	330137	338901	29.3
ibm10	1595	27508	16.7	newblue2	441586	465219	13.8
ibm11	1497	27477	20.0	newblue3	494123	552199	15.3
ibm12	1233	26320	20.6	newblue4	646219	637051	34.2
ibm13	954	27011	20.0	newblue5	1233154	1284251	25.4
ibm14	1635	43062	20.0	newblue6	1255135	1288443	40.7
ibm15	1412	52779	20.1				
ibm16	1091	47821	20.0				
ibm17	1442	56517	20.0				
ibm18	943	42200	20.0				

TABLE II
EXPERIMENTAL RESULTS ON IBM-HB+ BENCHMARK (mPL6 AND SCAMPI)

IBM-HB+	mPL6 + NTUplace-DP			SCAMPI		
	ovl%	WL (10^7)	RT (min)	WL (10^7)	diff%	RT (min)
ibm01	1.42%	0.33	0.79	0.34	3.0%	1.03
ibm02	1.35%	0.74	4.29	0.8	8.1%	2.33
ibm03	1.12%	0.92	2.13	0.95	3.3%	1.74
ibm04	1.65%	1.12	4.60	1.23	9.8%	2.40
ibm06	0.89%	0.92	3.87	1.1	19.6%	2.83
ibm07	1.61%	1.66	3.30	1.57	-5.4%	1.65
ibm08	1.80%	2.06	7.43	2.05	-0.5%	3.14
ibm09	1.54%	2.11	6.96	2.22	5.2%	3.03
ibm10	8.60%	7.06	11.08	5.52	-21.8%	5.33
ibm11	1.97%	3.00	9.26	2.78	-7.3%	2.41
ibm12	1.47%	5.64	7.42	6.76	19.9%	6.77
ibm13	1.96%	3.91	9.70	4.22	7.9%	3.49
ibm14	1.83%	7.52	9.88	6.64	-11.7%	4.47
ibm15	1.62%	10.35	21.23	8.82	-14.8%	6.27
ibm16	1.98%	10.23	10.76	10.62	3.8%	5.11
ibm17	2.00%	16.32	17.40	15.27	-6.4%	6.43
ibm18	1.98%	9.10	15.68	7.78	-14.5%	3.21
average	2.05%				-0.1%	

E. Stopping Criterion

The percentage of cell overlap OVL is used as a stopping criterion. For most circuits, OVL $\leq 10\%$ is small enough to be removed by the detailed placer. For the circuits with large module size variation (e.g., IBM-HB+ [11]), the stopping criterion can be set to OVL $\leq 3\%$ to remove more overlaps in the global placement phase.

VI. EXPERIMENTAL RESULTS

To solve problem (16) defined in Section II-B4, we implemented the gradient computation with both the quadratic penalty and augmented Lagrangian methods in a multilevel framework known as mPL6 [3].

The benchmarks used in the experiments include IBM-HB+ [11] and the modified ISPD'05 and ISPD'06 placement contest benchmarks [8], [9], which are summarized in Table I. The number of movable cells (including macros), the number of nets, and the percentage of white space are listed in the table.

The first experiment was performed on the IBM-HB+ benchmark [11], which consists of hard instances with large module size variation. The analysis in our Appendix I will show that, under such conditions, the exact smoothed density gradient computation is superior to the heuristic used in mPL6.

Authorized licensed use limited to: IEEE Xplore. Downloaded on December 17, 2008 at 14:51 from IEEE Xplore. Restrictions apply.

TABLE III
EXPERIMENTAL RESULTS ON IBM-HB+ BENCHMARK (OUR METHODS)

IBMHB+	Quadratic Penalty					Augmented Lagrangian				
	ovl%	WL(10^7)	diff%	RT(min)	diff%	ovl%	WL(10^7)	diff%	RT(min)	diff%
ibm01	1.34%	0.30	-9.09%	0.49	-38.0%	3.86%	0.28	-15.15%	0.55	-30.4%
ibm02	1.08%	0.61	-17.57%	1.48	-65.5%	1.58%	0.58	-21.62%	1.25	-70.9%
ibm03	1.10%	0.86	-6.52%	1.16	-45.5%	1.67%	0.82	-10.87%	1.12	-47.4%
ibm04	1.28%	1.06	-5.36%	1.54	-66.5%	1.87%	1.01	-9.82%	1.64	-64.3%
ibm06	0.87%	0.87	-5.43%	1.17	-69.8%	1.27%	0.80	-13.04%	0.92	-76.2%
ibm07	1.62%	1.58	-4.82%	2.24	-32.1%	2.41%	1.56	-6.02%	1.98	-40.0%
ibm08	1.02%	1.85	-10.19%	2.11	-71.6%	1.55%	1.71	-16.99%	1.73	-76.7%
ibm09	1.54%	1.61	-23.70%	1.83	-73.7%	2.27%	1.61	-23.70%	1.64	-76.4%
ibm10	1.57%	4.38	-37.96%	3.72	-66.4%	2.38%	3.96	-43.91%	2.96	-73.3%
ibm11	1.64%	2.75	-8.33%	2.68	-71.1%	2.40%	2.58	-14.00%	2.05	-77.9%
ibm12	1.47%	4.86	-13.83%	3.14	-57.7%	2.18%	4.85	-14.01%	2.22	-70.1%
ibm13	1.60%	3.56	-8.95%	2.29	-76.4%	2.42%	3.47	-11.25%	2.05	-78.9%
ibm14	1.41%	6.70	-10.90%	3.90	-60.5%	2.72%	6.50	-13.56%	3.59	-63.7%
ibm15	1.05%	8.12	-21.55%	5.17	-75.6%	1.66%	8.01	-22.61%	4.86	-77.1%
ibm16	1.99%	9.55	-6.65%	4.45	-58.6%	3.00%	9.37	-8.41%	4.46	-58.6%
ibm17	1.76%	14.98	-8.21%	6.19	-64.4%	2.97%	14.86	-8.95%	5.24	-69.9%
ibm18	1.99%	8.35	-8.24%	4.17	-73.4%	2.98%	7.78	-14.51%	4.06	-74.1%
average	1.4%		-12.19%		-62.8%	2.3%		-15.79%		-66.2%

TABLE IV
EXPERIMENTAL RESULTS ON THE MODIFIED ISPD'05 AND ISPD'06 BENCHMARKS

modified ISPD'05 ISPD'06	mPL6 + NTUplace-DP				Augmented Lagrangian + NTUplace-DP					
	WL (x 10^8)		RT (hour)		WL (x 10^8)			RT (hour)		
	global	final	global	total	global	final	diff%	global	total	diff%
adaptec2	0.84	0.86	0.72	0.77	0.82	0.81	-5.40%	0.79	0.83	6.80%
adaptec4	1.72	1.70	3.67	3.79	1.60	1.56	-8.40%	2.34	2.48	-34.50%
bigblue1	1.02	0.99	0.69	0.73	0.97	0.94	-5.60%	0.92	0.95	30.20%
bigblue2	1.19	1.14	4.16	5.02	1.10	1.07	-6.30%	2.89	3.55	-29.20%
bigblue3	3.36	3.24	5.73	6.18	3.48	3.36	3.60%	5.60	6.18	0.00%
bigblue4	7.94	7.73	7.12	7.84	7.99	7.62	-1.30%	8.48	9.15	16.70%
adaptec5	3.14	3.03	4.13	4.31	3.35	3.20	5.70%	3.89	4.08	-5.50%
newblue1	0.65	0.63	0.99	1.07	0.63	0.61	-2.50%	1.13	1.21	12.80%
newblue2	1.86	1.80	3.27	3.42	1.93	1.82	1.20%	2.89	3.07	-10.10%
newblue3	2.46	2.53	1.71	2.81	2.41	2.40	-5.00%	3.39	4.21	49.90%
newblue4	2.30	2.23	3.40	3.58	2.10	2.03	-9.00%	1.97	2.11	-40.90%
newblue5	4.08	3.92	5.95	6.36	3.98	3.80	-3.20%	5.46	5.83	-8.40%
newblue6	4.46	4.24	5.67	6.22	4.33	4.12	-2.80%	5.26	5.71	-8.20%
average							-3.00%			-1.60%

The placement results of mPL6, SCAMPI [11], the quadratic penalty method, and the augmented Lagrangian method are shown in Tables II and III. The data of SCAMPI are obtained from [11], but the other global placements are fed into the detailed placer of NTUplace-DP [4] for the final placement. The overlap-free condition was verified for the final placement. The column "ovl%" indicates the percentage of overlap after global placement. The columns "WL" and "RT" are the total wire length and the total run time for the whole placement, respectively. The column "diff%" is the relative difference compared to mPL6. The results show that the methods integrated with the exact gradient computation shorten the run time by almost 70% and improve the wire length by about 15%. These results reveal that, for the circuits with large module size variation, the exact gradient computation leads to faster convergence and better placement quality.

Moreover, this experiment also shows that the augmented Lagrangian method performs better than the quadratic penalty method, both in terms of wire length quality and run time. Although the average overflow after the global placement of the augmented Lagrangian method is slightly higher than the quadratic penalty method, the result shows that, within such small overflow, the global placement of the augmented Lagrangian method is indeed better because it leads to better final placement.

The second experiment was performed on the modified ISPD'05 [8] and ISPD'06 [9] benchmarks. The original benchmarks mostly consist of movable standard cells and fixed macros. We modified the benchmarks and made every object inside the placement region movable. In addition, the experiment is run for wire length minimization only instead of density-aware wire length minimization, as in the placement contests. The results of the modified benchmarks are shown in Table IV. Compared to the mPL6 results, the augmented Lagrangian method achieves an average of 3% shorter wire length, and it is as much as 9% shorter. The run time advantage of our method compared to mPL6 is not as significant as on IBM-HB+ benchmarks, because the macro size variations in the contest

Authorized licensed use limited to: IEEE Xplore. Downloaded on December 17, 2008 at 14:51 from IEEE Xplore. Restrictions apply.

benchmarks are not as large. We still achieve comparable or slightly better run time.

VII. Conclusion

In this paper, we introduced a general class of density smoothing operators and developed the theory and efficient algorithms for computing the gradient of density penalty functions in the nonlinear programming framework. This is the first time that the density-constrained placement problem with global smoothing technique can be solved exactly in the general nonlinear programming framework. We showed that such an approach supersedes the existing force-directed placement methods. The experiment on the IBM-HB+ benchmark shows the effectiveness of our technique.

Moreover, our approach with efficient computation of the density gradient shows promise in the application to the following problems: 1) 3-D placement with nonoverlap constraints and through silicon via density constraints and 2) thermal-aware placement with temperature constraints.

Appendix I

In this section, we perform further analysis on the theorems described in Section IV and show that they provide the algorithmic foundation to explain why the heuristics used in the existing force-directed methods work in the nonlinear programming framework. We also show why our approach of using the quadratic penalty method or augmented Lagrangian method with efficient gradient computation of the density penalty function is more general, stable, and theoretically sound.

A. Spreading Effect

If we solve the problem by the quadratic penalty method discussed in Section III-A, we have to solve $\|\nabla Q(\vec{x},\vec{y};\mu)\| \leq \tau$ multiple times, where $-\nabla WL(\vec{x},\vec{y}) - \nabla P_Q(\vec{x},\vec{y};\mu)$ is the steepest descent direction. The term $-\nabla WL(\vec{x},\vec{y})$ is the direction to reduce wire length, and the term $-\nabla P_Q(\vec{x},\vec{y};\mu)$ has a spreading effect that leads to a density feasible solution.

It can be proven that there exists a smoothing operator such that the smoothed density $\widehat{D}$ satisfies $\widehat{D}=\widehat{D}_H$, where $\widehat{D}_H$ is the smoothed density by the Helmholtz equation defined in Section II-B-1. For small-size modules, the spreading effect $-\nabla P_Q(\vec{x},\vec{y};\mu)$ for the kth module in x-direction can be approximated as follows:

$$
\begin{aligned}
&-\left(\widehat{D}(\vec{x},\vec{y}) - \widehat{I}\right) \odot \frac{\partial \widehat{D}(\vec{x},\vec{y})}{\partial x_k} \\
&\quad = -\int_{y_k-h_k/2}^{y_k+h_k/2} \left(\widehat{\widehat{D}}(\vec{x},\vec{y}) - \widehat{\widehat{I}}\right)(x_k + w_k/2, \nu')d\nu' \\
&\qquad + \int_{y_k-h_k/2}^{y_k+h_k/2} \left(\widehat{\widehat{D}}(\vec{x},\vec{y}) - \widehat{\widehat{I}}\right)(x_k - w_k/2, \nu')d\nu' \\
&\quad \approx -h_k \left(\widehat{\widehat{D}}(\vec{x},\vec{y}) - \widehat{\widehat{I}}\right)(x_k + w_k/2, y_k) \\
&\qquad + h_k \left(\widehat{\widehat{D}}(\vec{x},\vec{y}) - \widehat{\widehat{I}}\right)(x_k - w_k/2, y_k) \\
&\quad \approx -h_k w_k \partial\left(\widehat{\widehat{D}}(\vec{x},\vec{y})\right)(x_k, y_{ki})/\partial u \\
&\quad \approx -h_k w_k \partial\left(\widehat{D}_H(\vec{x},\vec{y})\right)(x_k, y_k)/\partial u. \qquad (48)
\end{aligned}
$$

This last expression is exactly the force computation in mPL6 [3], where $\partial(\widehat{D}_H(\vec{x},\vec{y}))(x_k,y_k)/\partial u$ was approximated by a finite difference scheme, and the module area $h_k w_k$ was the multiple factor used in mPL6 as a heuristic. From the assumptions made to reach these approximations, we know that this heuristic may fail when the module size variation is large. Compared to the heuristic force computed in [3], the spreading effect of our method is more general and is able to handle both small and large modules.

B. Holding Effect

Hold forces are used in Kraftwerk [5], [14]. It is not emphasized in [5], but an accumulative force is used in addition to the current force based on density. This force is the same as the hold force in [14], which is for the stability and support of Engineering Change Order. By using our gradient computation with the augmented Lagrangian method, we also observe the holding effect in our method.

If we solve the problem by the augmented Lagrangian method discussed in Section III-B, the multiplier λ is updated by $\lambda^{(k)} = \lambda^{(k-1)} + \lambda^{(k-1)}(\widehat{D}(\vec{x}^{(k)},\vec{y}^{(k)}) - \widehat{I})$ at the kth iteration. Since $(\vec{x}^{(k)},\vec{y}^{(k)})$ is the solution from the $(k-1)$th iteration, we have $\|\nabla L_A(\vec{x}^{(k)},\vec{y}^{(k)},^{(k-1)};\lambda^{(k-1)})\| = 0$ and

$$
\begin{aligned}
&\left\|\nabla L_A\left(\vec{x}^{(k)},\vec{y}^{(k)},\lambda^{(k-1)};\mu^{(k-1)}\right)\right\| \\
&= \left\|\nabla WL\left(\vec{x}^{(k)},\vec{y}^{(k)}\right) + \nabla P_A\left(\vec{x}^{(k)},\vec{y}^{(k)},\lambda^{(k-1)};\mu^{(k-1)}\right)\right\| \\
&= \Big\|\nabla WL\left(\vec{x}^{(k)},\vec{y}^{(k)}\right) \\
&\qquad + \left(\lambda^{(k-1)} + \mu^{(k-1)}\left(\widehat{D}\left(\vec{x}^{(k)},\vec{y}^{(k)}\right) - \widehat{I}\right)\right) \\
&\qquad \odot \nabla\, \widehat{D}\left(\vec{x}^{(k)},\vec{y}^{(k)}\right)\Big\| \\
&= \left\|\nabla WL\left(\vec{x}^{(k)},\vec{y}^{(k)}\right) + \lambda^{(k)} \odot \nabla\, \widehat{D}\left(\vec{x}^{(k)},\vec{y}^{(k)}\right)\right\|. \qquad (49)
\end{aligned}
$$

Therefore, $-\lambda^{(k)} \odot \nabla\, \widehat{D}(\vec{x}^{(k)},\vec{y}^{(k)}) = \nabla WL(\vec{x}^{(k)},\vec{y}^{(k)})$.

In Kraftwerk, the hold force at the kth iteration is defined as $F_{\text{hold}}^{(k)} = \nabla WL(\vec{x}^{(k)},\vec{y}^{(k)})$, where $(\vec{x}^{(k)},\vec{y}^{(k)})$ is the solution

Authorized licensed use limited to: IEEE Xplore. Downloaded on December 17, 2008 at 14:51 from IEEE Xplore. Restrictions apply.

from the $(k-1)$th iteration. It is clear that this holding force is approximately $-\lambda^{(k)} \odot \nabla \widehat{D}(\vec{x}^{(k)}, \vec{y}^{(k)})$.

Moreover, the steepest descent direction of the augmented Lagrangian function at the kth iteration is

$$-\nabla WL(\vec{x},\vec{y}) - \lambda^{(k)} \odot \nabla \widehat{D}(\vec{x},\vec{y}) - \nabla P_Q\left(\vec{x},\vec{y};\mu^{(k)}\right) \quad (50)$$

where $-\nabla P_Q(\vec{x},\vec{y};\mu^{(k)})$ is the spreading effect, as analyzed in the previous section, and $-\lambda^{(k)} \odot \nabla \widehat{D}(\vec{x},\vec{y})$ is approximately the holding force as in Kraftwerk at the starting point $(\vec{x}^{(k)}, \vec{y}^{(k)})$. The difference between Kraftwerk and our method is that the holding force is constant during each iteration but $-\lambda^{(k)} \odot \nabla \widehat{D}(\vec{x},\vec{y})$ is not. The study of its effect will be left for future work.

By using the augmented Lagrangian method, we also separate the two kinds of effects, where the holding effect is related to the Lagrangian multipliers and the spreading effect is related to the penalty parameters.

APPENDIX II

A. Hölder's Inequality [15]

Suppose $1 < p < \infty$ and $p^{-1} + q^{-1} = 1$. If f and g are measurable functions, then $\|fg\|_1 \le \|f\|_p \|g\|_q$.

In particular, if $f \in L^p$ and $g \in L^q$, then $fg \in L^1$.

B. The Fubini–Tonelli Theorem [15]

Define the x-section f_x and y-section f_y by $f_x(y) = f_y(x) = f(x,y)$, and denote the set of measurable functions from the set X to $[0,\infty]$ by $L^+(X)$. Suppose (X, M, μ) and (Y, N, ν) are σ-finite measure spaces.

(Tonelli) If $f \in L^+(X \times Y)$, then the functions $g(x) = \int f_x(y)\, d\nu(y)$ and $h(y) = \int f_y(x)\, d\mu(x)$ are in $L^+(X)$ and $L^+(Y)$, respectively, and

$$\int f(x,y)\, d(\mu(x) \times \nu(y)) = \int \left[\int f(x,y)\, d\nu(y)\right] d\mu(x) = \int \left[\int f(x,y)\, d\mu(x)\right] d\nu(y) \quad (51)$$

(Fubini) If $f \in L^1(\mu \times \nu)$, then $f_x \in L^1(\nu)$ for almost every $x \in X$, and $f_y \in L^1(\mu)$ for almost every $y \in Y$; the almost-everywhere-defined functions $g(x) = \int f_x(y)\, d\nu(y)$ and $h(x) = \int f_y(x)\, d\mu(x)$ are in $L^1(\mu)$ and $L^1(\nu)$, respectively, and (51) holds.

APPENDIX III

Here we explain step (36) for G with one singularity. This extends easily for any finite number of singularities. Suppose that a singularity exists at point (u^*, ν^*) where $u^* \in (x_k - w_k/2, x_k + w_k/2)$ and $\nu^* \in (y_k - h_k/2, y_k + h_k/2)$.

We can rewrite the function at step (35) as

$$\frac{\partial}{\partial x_k}\left[\lim_{n\to\infty}(u,\nu,x_k,y_k)\right] \quad (52)$$

where

$$f_n(u,\nu,x_k,y_k) = \int_{x_k-w_k/2}^{u^*-1/n}\int_{y_k-h_k/2}^{y_k+h_k/2} G(u,\nu,u',\nu')d\nu' du' + \int_{u^*+1/n}^{x_k+w_k/2}\int_{y_k-h_k/2}^{y_k+h_k/2} G(u,\nu,u',\nu')d\nu' du'. \quad (53)$$

Note that each f_n is differentiable with respect to x_k on some interval; call it $[A_k, B_k]$. By the Leibniz integral rule we have

$$\frac{\partial}{\partial x_k} f_n(u,\nu,x_k,y_k) = \int_{y_k-h_k/2}^{y_k+h_k/2} G\left(u,\nu,x_k + w_k/2,\nu'\right) d\nu' - \int_{y_k-h_k/2}^{y_k+h_k/2} G\left(u,\nu,x_k - w_k/2,\nu'\right) d\nu'. \quad (54)$$

Note that since each $(\partial/\partial x_k) f_n(u,\nu,x_k,y_k)$ is independent of the index n, the sequence converges uniformly on $x_k \in [A_k, B_k]$

$$\lim_{n\to\infty}\left[\frac{\partial}{\partial x_k} f_n(u,\nu,x_k,y_k)\right] = \int_{y_k-h_k/2}^{y_k+h_k/2} G(u,\nu,x_k + w_k/2,\nu')d\nu' - \int_{y_k-h_k/2}^{y_k+h_k/2} G(u,\nu,x_k - w_k/2,\nu')d\nu'. \quad (55)$$

We also know that for some $x_k \in [A_k, B_k]$ (namely the original x_k), $f_n(u,\nu,x_k,y_k)$ converges

$$\lim_{n\to\infty} f_n(u,\nu,x_k,y_k) = \int_{x_k-w_k/2}^{x_k+w_k/2}\int_{y_k-h_k/2}^{y_k+h_k/2} G(u,\nu,u',\nu')d\nu' du' \quad (56)$$

since $G \in L^2 \subset L^1$ in the integrated region.

From [16] we have the result that if f_n is a sequence of functions differentiable on $[A, B]$ and such that $f_n(x_o)$ converges for some point $x_o \in [A, B]$, then f_n converges uniformly to a

Authorized licensed use limited to: IEEE Xplore. Downloaded on December 17, 2008 at 14:51 from IEEE Xplore. Restrictions apply.

function f, and $f'(x) = \lim_{n\to\infty} f'_n(x)$. Invoking this result in the current problem, we obtain

$$\frac{\partial}{\partial x_k}\left[\lim_{n\to\infty} f_n(u,\nu,x_k,y_k)\right] = \lim_{n\to\infty}\left[\frac{\partial}{\partial x_k} f_n(u,\nu,x_k,y_k)\right]. \tag{57}$$

Substituting (55) on the right hand side, we obtain step (36).

ACKNOWLEDGMENT

The authors would like to thank Prof. T. Chan, Prof. L. Vandenberghe, and J. Lee for many inspiring discussions.

REFERENCES

[1] K. Arrow, L. Huriwicz, and H. Uzawa, *Studies in Nonlinear Programming*. Stanford, CA: Stanford Univ. Press, 1958.
[2] T. Chan, J. Cong, and K. Sze, "Multilevel generalized force-directed method for circuit placement," in *Proc. Int. Symp. Phys. Des.*, 2005, pp. 185–192.
[3] T. Chan, J. Cong, J. Shinnerl, K. Sze, and M. Xie, "mPL6: Enhancement multilevel mixed-size placement with congestion control," in *Modern Circuit Placement*, G.-J. Nam and J. Cong, Eds. New York: Springer-Verlag, 2007.
[4] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang, "A high-quality mixed-size analytical placer considering preplaced blocks and density constraints," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2006, pp. 187–192.
[5] H. Eisenmann and F. M. Johannes, "Generic global placement and floorplanning," in *Proc. 35th Annu. Conf. Des. Autom.*, 1998, pp. 269–274.
[6] A. B. Kahng, S. Reda, and Q. Wang, "Architecture and details of a high quality, large-scale analytical placer," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2005, pp. 891–898.
[7] D. G. Luenberger, *Optimization by Vector Space Methods*. Hoboken, NJ: Wiley, 1969.
[8] G.-J. Nam, C. J. Alpert, P. Villarrubia, B. Winter, and M. Yildiz, "The ISPD2005 placement contest and benchmark suite," in *Proc. Int. Symp. Phys. Des.*, 2005, pp. 216–220.
[9] G.-J. Nam, "ISPD 2006 placement contest: Benchmark suite and results," in *Proc. Int. Symp. Phys. Des.*, 2006, p. 167.
[10] W. C. Naylor, R. Donelly, and L. Sha, "Non-linear optimization system and method for wire length and delay optimization for an automatic electric circuit placer," U.S. Patent 6 301 693, Oct. 9, 2001.
[11] A. N. Ng, I. L. Markov, R. Aggarwal, and V. Ramachandran, "Solving hard instances of floorplacement," in *Proc. Int. Symp. Phys. Des.*, 2006, pp. 170–177.
[12] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York: Springer-Verlag, 2006.
[13] A. D. Polyanin, *Handbook of Linear Partial Differential Equations for Engineers and Scientists*. London, U.K.: Chapman & Hall, 2002.
[14] P. Spindler, U. Schlichtmann, and F. M. Johannes, "Kraftwerk2—A fast force-directed quadratic placement approach using an accurate net model," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 8, pp. 1398–1411, Aug. 2008.
[15] G. B. Folland, *Real Analysis: Modern Techniques and Their Applications*, 2nd ed. New York: : John Wiley & Sons, 1999.
[16] W. Rudin, *Principles of Mathematical Analysis*, 3rd ed. New York: McGraw-Hill, 1976.

Jason Cong (S'88–M'90–SM'96–F'00) received the B.S. degree in computer science from Peking University, Beijing, China, in 1985 and the M.S. and Ph.D. degrees in computer science from the University of Illinois, Urbana, in 1987 and 1990, respectively.

He is currently a Professor and the Chairman with the Department of Computer Science, University of California, Los Angeles (UCLA), where he is also the Codirector of the VLSI CAD Laboratory. His research interests include computer-aided design of very large scale integration circuits and systems, design and synthesis of system-on-a-chip, programmable systems, novel computer architectures, nanosystems, and highly scalable algorithms. He has published over 250 research papers and led over 30 research projects in these areas. He has served on the Technical Advisory Board of a number of EDA and silicon IP companies, including Atrenta, eASIC, Get2Chip, Magma Design Automation, and Ultima Interconnect Technologies. He was the Founder and President of Aplus Design Technologies, Inc., until it was acquired by Magma Design Automation, in 2003. Currently, he serves as the Chief Technology Advisor of Magma and AutoESL Design Technologies, Inc.

Dr. Cong received a number of awards and recognitions, including the Ross J. Martin Award for Excellence in Research from the University of Illinois in 1989, the National Science Foundation Young Investigator Award in 1993, the Northrop Outstanding Junior Faculty Research Award from UCLA in 1993, the Association for Computing Machinery (ACM)/Special Interest Group on Design Automation Meritorious Service Award in 1998, and the Semiconductor Research Corporation Technical Excellence Award in 2000. He also received four Best Paper Awards selected for the 1995 IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN, the 2005 International Symposium on Physical Design, the 2005 ACM Transaction on Design Automation of Electronic Systems, and the 2008 International Symposium on High Performance Computer Architecture, respectively.

Guojie Luo (S'08) received the B.S. degree in computer science from Peking University, Beijing, China, in 2005 and the M.S. degree in computer science from the University of California, Los Angeles, in 2008, where he is currently working toward the Ph.D. degree in computer science in the Department of Computer Science.

His current research interests include physical design algorithms, 3-D integration technology, and nonlinear optimization.

Eric Radke received the B.S. degree in mathematics from Case Western Reserve University in 2004 and the M.A. degree in applied mathematics from the University of California, Los Angeles, where he is currently working toward his Ph.D. degree in applied mathematics in the Department of Mathematics.

His current research interests include nonlinear optimization, VLSI physical design, and numerical analysis of partial differential equations.

Authorized licensed use limited to: IEEE Xplore. Downloaded on December 17, 2008 at 14:51 from IEEE Xplore. Restrictions apply.