An Interior Point (Karmarkar) Project for Solving the Global Routing Problem

NE 112

Linear Algebra with Numerical Applications

Fall 2005

P. Lee, R. Swaminathan

December 3, 2005

Contents

1 Theoretical Introduction

1.1 Background

The purpose of this project is to serve as an introduction to the *Interior Point Method* for solving a very large scale integration (VLSI) circuit layout problem; that is, the *global routing* problem. The global routing problem is one that commonly arises in computer chip circuitry. Modern computer chips often have hundreds and thousands, if not millions, of modules that need to be laid out and connected using pins and wires. This network of wires is crucial for a timely transfer of information from one module to another.

In designing optimal chips for best performance, we ask ourselves some relevant questions:

1. Given the inherent limitations of the chip, namely space and bandwidth restrictions, is it possible to connect *all* my modules in an efficient manner?
2. If yes, what layout will result in the shortest length of wire used?
3. If no, what percentage of modules can I wire successfully?

Wire length is an important consideration as the wire resistance, and in turn, the power dissipated via the Joule effect is proportional to the total length of wire used. This can lead to tremendous cooling problems resulting in "hot-spots" therefore causing the chip to cease normal functionality. The efficiency of the chip is also decreased substantially as a large proportion of power input is discarded as waste heat.

The Interior Point Method is used to determine the best possible wire combination keeping in mind bandwidth and wire length considerations. If a thorough solution is unobtainable, the interior point provides us with information needed to assess what percentage of wiring is indeed feasible.

1.2 LP Modeling and Methodology

Consider the 2 $\times$ 3 grid problem given in Figure 1. Suppose that we want to connect the three modules A, B and C by a wire using horizontal and vertical segments.

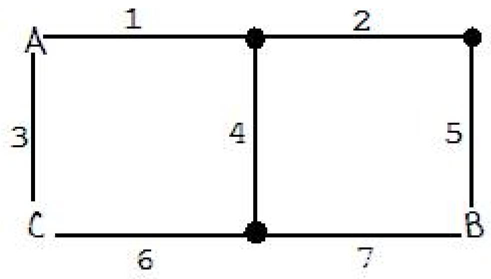

Figure 1: A 2 $\times$ 3 example grid

In this case, we want a connection from module A to module B, module B to module C and module C to module A. We call each of these connections a *net*. The obvious solution is to connect all modules directly with one path. Of course, this would not be the case if certain segments had capacities less than 2. Consider the case where all segments have a capacity of one. The problem becomes less direct now. Segment capacities are reached and pathway congestion occurs preventing us from using the most direct route. Thus, an alternate pathway must be found by re-routing through one or more connectors.

It is evident that in more complicated problems involving hundreds and thousands of modules, the most effective solution to the problem cannot be ascertained using human logic just shown. Thus, a more quantitative analysis must be applied.

First, all possible pathways to connect Module X to Module Y must be presented. Each module pathway may have several possibilities or *trees*, denoted by y_i. For each possible pathway, specific segments are used as shown below.

Net	Connection	y_1	y_2	y_3	y_4
1	A to B	(1,2,5)	(3,6,7)	(1,4,7)	–
2	B to C	(7,6)	(7,4,1,3)	(5,2,1,3)	(5,2,4,6)
3	C to A	(3)	(6,4,1)	(6,7,5,2,1)	–

In this notation, p_{11} refers to the first connection (y_1) in net 1, p_{32} refers to the second connection (y_2) in the third net and so on. In other words, given p_{ij}, i is the net number and j is the tree choice (y_j). $p_{ij} = 1$ means that the corresponding path is picked, 0 otherwise.

The first consideration is that for each given net, we need to choose one and only one tree. The constraint is expressed as

$$\sum_{j=1}^{4} p_{ij} \leq 1, i \in \{1,2,3\}$$

or,

$$\begin{aligned} p_{11} + p_{12} + p_{13} &\leq 1 \\ p_{21} + p_{22} + p_{23} + p_{24} &\leq 1 \\ p_{31} + p_{32} + p_{33} &\leq 1 \end{aligned} \tag{1}$$

The second consideration deals with the fact that pathway values cannot be negative. The constraints can be expressed as $p_{ij} \in \{0,1\}$, but for the purpose of our example (to allow for iterative processes to converge to the solution), we let

$$p_{ij} \geq 0, i \in \{1,2,3\}, j \in \{1,2,3,4\} \tag{2}$$

The third consideration is that when two or more trees are picked, the number of wires running through any given segment must not exceed its capacity denoted by c_i.

Paths through segment $i \leq c_i$, or

$$\begin{aligned}
p_{11} + p_{13} + p_{22} + p_{23} + p_{32} + p_{33} &\leq c_1 \\
p_{11} + p_{23} + p_{24} + p_{33} &\leq c_2 \\
p_{12} + p_{22} + p_{23} + p_{13} &\leq c_3 \\
p_{13} + p_{22} + p_{24} + p_{32} &\leq c_4 \\
p_{11} + p_{23} + p_{24} + p_{33} &\leq c_5 \\
p_{12} + p_{21} + p_{24} + p_{32} + p_{33} &\leq c_6 \\
p_{12} + p_{13} + p_{21} + p_{22} + p_{33} &\leq c_7
\end{aligned} \tag{3}$$

For this example $c_i = 1$ for all $i = 1, 2, \ldots, 7$.

The fourth and last consideration is the *minimization of wirelength*. This is done by giving more weightage to preferred paths. Typically longer paths are inefficient and increase the total resistance of the circuit. Therefore, shorter paths are generally favored. Let

$$\text{wiremax} = \text{maximum wirelength for all paths}$$

An appropriate measure of wirelength is

$$b_{ij} = (\text{wiremax} + 1) \text{ - wirelength of tree } p_{ij}$$

where b_{ij} is the *weighting factor*. Shorter wirelengths have larger weighing factors to give them more preference and vice versa. For example, we can calculate b_{11} by realizing that wiremax $= 5$ and so $b_{11} = (5 + 1) - (3) = 3$.

Maximizing the following function results in picking up trees with the shortest wirelengths:

$$3p_{11} + 3p_{12} + 3p_{13} + 4p_{21} + 2p_{32} + 2p_{23} + 2p_{24} + 5p_{31} + 3p_{32} + p_{33} \tag{4}$$

The weighting factor is not based on wirelength only; many other factors may be included too. In complicated scenarios, multi-metal layered chips may have favorable metal levels through which a wire is best routed through. The chip designer himself may have special considerations that may have to be incorporated in choosing a more preferred tree. For the sake of this illustration, we will use wirelengths only to determine tree preference.

Using (1), (2), (3) and (4), we have the following **linear integer programming** (LP) representation of the global routing problem:

$$\text{MAX} \quad 3p_{11} + 3p_{12} + 3p_{13} + 4p_{21} + 2p_{32} + 2p_{23} + 2p_{24} + 5p_{31} + 3p_{32} + p_{33}$$

$$\begin{aligned}
p_{11} + p_{12} + p_{13} &\leq 1 \\
p_{21} + p_{22} + p_{23} + p_{24} &\leq 1 \\
p_{31} + p_{32} + p_{33} &\leq 1 \\
p_{11} + p_{13} + p_{22} + p_{23} + p_{32} + p_{33} &\leq c_1 \\
p_{11} + p_{23} + p_{24} + p_{33} &\leq c_2
\end{aligned} \tag{5}$$

$$
\begin{aligned}
p_{12}+p_{22}+p_{23}+p_{13} &\leq c_3 \\
p_{13}+p_{22}+p_{24}+p_{32} &\leq c_4 \\
p_{11}+p_{23}+p_{24}+p_{33} &\leq c_5 \\
p_{12}+p_{21}+p_{24}+p_{32}+p_{33} &\leq c_6 \\
p_{12}+p_{13}+p_{21}+p_{22}+p_{33} &\leq c_7
\end{aligned}
$$

$$p_{ij} \geq 0, \text{ for } i \in \{1,2,3\} \text{ and } j \in \{1,2,3,4\}$$

In addition, $c_i = 1$, for $i = 1, 2, \ldots, 7$ for the sake of example.

(5) is a system of linear *inequalities* and therefore cannot be solved by conventional reduction or elimination methods. We need a more sophisticated method to find an optimal solution that fits within the constraints specified.

2 The Interior Point Algorithm

Starting from Karmarkar's epoch-making paper [1] in 1984, research on interior-point methods (IPMs) has dominated the field of linear optimization for more than fifteen years, and over three thousand papers have been published relating to IPMs. The implementation of the Interior Point Algorithm takes a different approach than from the Simplex Method [2] in that it starts with a point known as the initial guess inside the n-dimensional linearly constrained space and constructs ellipsoids inside this feasible region. Following this, a projection is calculated in the direction of the optimal solution and the intersection of the projection with the ellipsoid's boundary becomes the updated guess point. The process is iterated through repeatedly until a close enough solution is obtained. This process will never actually converge to an answer because the ellipsoid constructed will never exactly touch a corner where the most optimal solution is to be found. The margin of error results due to this distance between the ellipsoid's boundary and the optimal solution.

The projection itself is based on a number of factors such as projection direction, correction angle, and step-length parameter γ, where $0 < \gamma < 1$. This parameter enables us to control how far a jump should be taken given a projected direction. Although it might seem that $\gamma = 1$ is the most optimal value, we need to keep in mind that the direction calculated initially may not point directly toward the optimal solution. A large value for γ can thus prematurely lead the interior point to a boundary causing it to "zig-zig", thereby increasing the number of iterations required. Choosing an optimal value for γ for a given LP problem will be explored in further detail in the following sections.

2.1 Pseudo Code

Step 1

The interior point method requires five input variables: A coefficient matrix and a capacity vector $\mathbf{A} \in \Re^{m \times n}$ and $\mathbf{b} \in \Re^m$ that correspond to the left and right hand side of (5) respectively; a vector $\mathbf{c}$ that corresponds to the coefficients of the objective function, in our case it is the function constituting weighted trees; a vector $\mathbf{x^0}$ of initial points that lies within the boundaries of the n-dimensional space, i.e. $\mathbf{A}\mathbf{x}^0 < \mathbf{b}$; and a step-length parameter γ.

Step 2

The interior point starts off by finding how far the initial guess is from each constraint.

$$\mathbf{v}^k \text{ (slack)} = \mathbf{b} - \mathbf{A}\mathbf{x}^k = \left[v_1^k, v_2^k, \cdots, v_m^k\right]^T$$

Since from step 1, $\mathbf{b} > \mathbf{A}\mathbf{x}^0$, $\mathbf{v}^k$ will always be positive. For each iteration the updated point $\mathbf{x}^0$ gets closer to the optimal value $\mathbf{x}$, and the slack becomes increasingly smaller. This correlates with the geometric explanation that as the ellipsoid drawn gets closer to the optimal corner in the n-dimensional space, the difference between the corner and the circle gets smaller.

Step 3

Once the ellipsoid has been defined, the direction of the projection is calculated. A diagonal matrix $\mathbf{D} \in \Re^{m\times m}$ is created first by inverting each slack and placing them across the main diagonal.

$$\mathbf{D}^k = \text{diag}\left[\frac{1}{\mathbf{v}_1^k}, \frac{1}{\mathbf{v}_2^k}, \cdots, \frac{1}{\mathbf{v}_m^k}\right]$$

Step 4

We use $\mathbf{D}_k$ to find the projection of the current guess onto the boundary of the ellipsoid. In doing so, we solve a system containing a quadruple product

$$\left(\mathbf{A}^T\mathbf{D}_k\mathbf{D}_k\mathbf{A}\right)\mathbf{dx} = \mathbf{c} \tag{6}$$

where $\mathbf{dx}$ is the directional matrix and $\mathbf{c}$ is the coefficient vector of the objective function. This is another linear system that needs solving, which will be dealt with later. The observation that the quadruple product $\mathbf{A}^T\mathbf{D}_k\mathbf{D}_k\mathbf{A}$ is always **symmetric** and **positive definite** plays a key role in one of the techniques used to solve (6).

Step 5

We then proceed to scale the projected direction vector $\mathbf{dx}$ as defined by the coefficient matrix $\mathbf{A}$.

$$\mathbf{dv} = -\mathbf{A}\mathbf{dx} = \left[(\mathbf{dv})_1, (\mathbf{dv})_2, \cdots, (\mathbf{dv})_m\right]^T$$

Step 6

This step takes the ratio of the slack and the corresponding scaled vector $\mathbf{dv}$, after which the maximum valid projected length is chosen. The step length α is given by

$$\alpha = \gamma \times \text{MAX}\left\{\frac{\mathbf{v}_i^k}{(\mathbf{dv})_i}, \text{ for } (\mathbf{dv})_i < 0\right\}$$

where $i = 1, 2, \ldots, m$.

We discard positive values in $(\mathbf{dv})_i$ as they point away from the optimal solution. As a result, the maximum of $\mathbf{dv}$ will yield the value closest to zero as our step length to be further scaled by γ.

New guesses or $\mathbf{x}$ values are calculated by

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha \mathbf{dx}$$

where $\alpha\mathbf{dx}$ is the projection (magnitude and direction) or the change in $\mathbf{x}$, a factor that leads to a better approximated interior point $\mathbf{x}^{k+1}$.

Step 7

The interior point method is an iterative process. For each iterative cycle, we get closer to the optimal solution giving rise to better $\mathbf{x}$ values. As the number of iterations k proceeds to infinity, $\mathbf{x}^{k+1}$ will equal $\mathbf{x}$. For simple problems, we may be satisfied with a reasonable level of accuracy identified by ϵ. The stopping criterion (specified relative to the previous iteration) is given by

$$\frac{|\mathbf{c}^T\mathbf{x}^{k+1} - \mathbf{c}^T\mathbf{x}^k|}{|\mathbf{c}^T\mathbf{x}^k| + \epsilon} \leq \epsilon$$

Typically, ϵ is of the order 10^{-6}; however, we can choose ϵ to be as small as we want depending on the level of accuracy desired.

3 A MATLAB Implementation

The complete MATLAB code for the Interior Point Algorithm can be found in Appendix A of this document.

In Step 4 of the interior point algorithm, we are required to calculate the direction of the projection vector by solving (6). Quite expectedly, there are several methods to approach this. We detail three methods used in our implementation of the interior method.

3.1 Cholesky Factorization

In MATLAB[1], an equation of the form $\mathbf{Ax} = \mathbf{b}$ can be conveniently solved using `mldivide` or the backslash operator. This examines the input matrix $\mathbf{A}$, performs a series of tests on it, and determines the best possible method to obtain the solution vector $\mathbf{x}$. As mentioned previously, the quadruple product $\mathbf{A}^T\mathbf{D}_k\mathbf{D}_k\mathbf{A}$ is symmetric and positive definite and thus makes it an ideal candidate to be used in a Cholesky factorization method. In our implementation, we programmed the factorization explicitly. In MATLAB, a pre-programmed function `chol` exists to obtain the factors of the quadruple product. We solve the system $\mathbf{Ax} = \mathbf{b}$ quite straightforwardly:

[1]Matrix Laboratory (Mathworks, Inc.)

```
        R = chol(A);
        x = R \ (R' \ b);
```

3.2 Gauss-Seidel Iterative Technique

The Gauss-Seidel method is an **iterative** technique which accepts an initial guess and iterates to obtain a better guess, much like the interior point method itself. A system of the form $\mathbf{Ax} = \mathbf{b}$ can be written as $(\mathbf{D} - \mathbf{L} - \mathbf{U})\mathbf{x} = \mathbf{b}$, where $\mathbf{D}$, $-\mathbf{L}$ and $-\mathbf{U}$ are respectively, elements on the main diagonal, lower triangular and upper triangular parts of the matrix $\mathbf{A}$. Re-arranging, we obtain

$$\mathbf{x}_{i+1} = (\mathbf{D} - \mathbf{L})^{-1}\mathbf{U}\mathbf{x}_i + (\mathbf{D} - \mathbf{L})^{-1}\mathbf{b} \tag{7}$$

The Gauss-Seidel method is computationally efficient and requires a minimum number of iterations because it uses updated values from $\mathbf{x}_1$ to $\mathbf{x}_{i-1}$ to compute $\mathbf{x}_i$. This process is incorporated into the matrix form (7) and does not have to be programmed separately.

In our implementation, the Gauss-Seidel routine always uses a vector of zeroes as an initial guess. An indefinite while loop is then used to iterate repeatedly until the value of $\mathbf{x}$ is within a desired degree of accuracy. This is indicated by the stopping criterion

$$\|\mathbf{y}\|_2 = \left(\sum_{i=1}^{n} \mathbf{y}_i^2\right)^{1/2} \leq \epsilon = 10^{-5}, \text{where } \mathbf{y} = \mathbf{Ax} - \mathbf{b}$$

upon which the MATLAB `break` statement is used to terminate the loop.

3.2.1 A von Neumann Polynomial Preconditioner

In attempting to solve a system of the form $\mathbf{Ax} = (\mathbf{A^T D_k^2 A})\mathbf{dx} = \mathbf{c}$, the input matrix $\mathbf{A}$ gets more complicated as the interior point method converges to the solution. In such cases, $\det(\mathbf{A}) \approx 0$ and the `gauss-seidel` routine takes a large number of iterations to converge. Sometimes, the number of iterations exceeds the maximum specified by us, thereby returning an error. The famous Hungarian-born mathematician John von Neumann presented a very fast matrix preconditioner for finding the approximate inverse matrix $\mathbf{A^{-1}}$ of the matrix $\mathbf{A}$. He did so by truncating the McLaurin series generalization for matrices and obtaining the following formula:

$$\mathbf{A^{-1}} \approx \mathbf{P} = \mathbf{D^{-1}}\left(\sum_{i=0}^{k} \left(\mathbf{CD^{-1}}\right)^i\right)$$

where $\mathbf{P}$ is the preconditioner matrix, and $\mathbf{D}$ and $-\mathbf{C}$ are the respective *diagonal* and *off-diagonal* elements of $\mathbf{A}$.

We therefore solve the system $\mathbf{PAx} = \mathbf{Pc}$ using the Gauss-Seidel technique which reduces the number of iterations significantly, especially for a close-to-singular matrix $\mathbf{A}$. The matrix $\mathbf{PA}$ is close to an identity matrix thereby simplifying the problem.

In our MATLAB implementation, $\mathbf{A^{-1}}$ was computed via a function.

```
function [P] = precondition(B, k)

    D = diag(diag(B));
    C = D - B;
    r = C * inv(D);
    inverse = 0;

    for j = 0:k
        inverse = inverse + r^j;
    end

    P = inv(D) * inverse;

end
```

where k is the number of terms to be used in the McLaurin series expansion for matrices for the following function

$$f(x) = \frac{1}{1-x}$$

given by

$$f(x) = \sum_{i=0}^{\infty} x^i = 1 + x^2 + x^3 + x^4 + \cdots$$

when

$$|x| < 1$$

In our `gauss-seidel` routine, we had triggers set-up to initiate preconditioning. However, once preconditioning was triggered, we ensured that this stayed the same throughout.

3.3 Preconditioned Conjugate Gradient Method

The actual theory beneath the preconditioned conjugate gradient (PCG) method is outside the scope of our understanding at this stage, but the general idea has been understood by a casual reading of parts of [3]. Our interpretation is summarized in Appendix C.

Unfortunately, the PCG method is far beyond our understanding to actually code, so we use the pre-programmed MATLAB function to solve the equation $\mathbf{proj} \cdot \mathbf{dx} = \mathbf{c}$:

```
% Way 5 - Preconditioned Conjugate Gradient Method
[dx, flag] = pcg(proj, c, eps, 30);
```

where `eps` is the error tolerance desired, 30 is the maximum number of iterations and `flag` is an argument returned by the in-built function `pcg`.

4 Results and Evaluation

We tested our code against two medium-sized routing problems. We will look at each problem individually.

4.1 Routing Problem 1

In problem #1, we deal with a 12 net (wire) consisting of 25 modules on a 6×6 grid. The input matrix $\mathbf{A}$ is reasonably large of dimensions 31×19. The matrix is not particularly sparse, but the interior point method was successful in converging to the solution in as less as **ten iterations**.

The linear integer programming representation of this problem and it's corresponding output is given in Appendix B. After we obtained this solution, we asked ourselves "How sensible is this answer?" We reasoned that our results made sense since the maximizing function picks the paths with coefficients 5, observing that the constraints for any given path do not overlap. In other words, there are no two constraints that involve the same path and the problem is *separately maximized* by the objective function. It can be shown that this is indeed the case by verifying the solution obtained. The numerical value of the function being maximized appears under the dashed line.

4.2 Routing Problem 2

In problem #2, we solved a 10 net (wire) problem. The input matrix $\mathbf{A}$ was fairly large of dimensions 50×20. The interior point algorithm successfully converged to the right solution in as less as **eleven iterations**.

The linear integer programming representation of this problem and it's corresponding output is given in Appendix B. The problem is structured is such a way that maximizing the objective function will simply pick the wires with greatest weights. In other words, the problem is once again separately maximized by the objective function. The solution obtained may be verified against this observation. The numerical value of the objective function appears under the dashed line.

4.3 Inadequate Constraints

During our initial stages of testing of Routing Problem 1, we forgot to input the last few constraints, namely $y_j \geq 0$ for $j = 1, 2, \ldots, 19$ into our input matrix $\mathbf{A}$. Doing so allowed the y_j variables to assume negative values. When attempting to solve such inconsistent systems, the `gauss_seidel` routine returns a warning:

```
Warning:  Matrix is singular, close to singular or badly scaled to
working precision.
```

On a similar note, the `cholesky` factorization method would return an error:

```
???  Error using ==> chol
Matrix must be positive definite.
```

Only MATLAB's built-in `PCG` function would return a result. However, this result comprised of negative values for y_j, which although is the solution to the system (the objective function was indeed maximized), is not physically meaningful.

The problem was corrected for by adding the extra constraints into the input matrix.

4.4 Parameter Analysis

In Section 2, we described the trait in which the interior point "zig-zags" toward the solution upon getting too close to a boundary. This leads to an increase in the number of iterations. One method to solve this would be to make the parameter γ—a parameter that controls the step length—a function of k, the iteration counter. This would lead to taking large step lengths in the beginning but smaller step lengths as we approach the solution.

We found that the number of outer interior point iterations k_{ipm} and the maximum number of inner `gauss-seidel` iterations k_{gs} changed as a function of γ. The results for Routing Project 1 are tabulated in Table 1. A starred number indicates that the number of iterations had crossed the limit set by the `gauss-seidel` routine.

γ	k_{ipm}	k_{gs} (max)
0.9999	5	38000*
0.999	6	38000*
0.99	7	31177
0.90	10	314
0.85	12	141
0.80	12	81
0.75	13	53
0.70	13	38
0.60	16	29
0.50	20	29
0.40	26	29

Table 1: Parameter analysis for Routing Project 1

Guessing k_{max} before determining γ, it turns out, is not as straightforward, so we resorted to predicting it as the interior point algorithm was running and we had some values from the first few iterations. We decided to come up with a **feedback control system** that would predict k_{max} using these values, change γ accordingly and lower the number of iterations. This process would repeat itself every five or so iterations. The key here is that all of this would happen *as the algorithm was running*. However, the implementation of this technique got a bit hairy towards the end leading us to abandon it. Furthermore, this method is not substantially useful for simple problems: the reduction in iterations is only about five or less. For more complicated problems, a thorough implementation of this technique seems feasible.

5 Conclusion

This project was a grand success. All four examples that our code was tested against worked on the first attempt without major problems. Two minor problems encountered were a) a division by zero error in Step 6 of the interior-point algorithm and b) negative results error because of omitting the extra required constraints $y_j \geq 0$. Both of these errors were fixed without substantial effort. In general, all requirements stated in the handed booklet were met without significant problems.

An interesting observation made during the course of our project was that the Cholesky factorization method worked very well for relatively large and non-complex matrices; however, as the interior-point method converged toward the optimal solution, the quadruple product in (6) increased dramatically in both size and complexity, and iterative techniques like Gauss-Seidel proved to be more computationally speedier. These observations led us to theorize a solution where the code automatically switched between using Cholesky and Gauss-Seidel during execution to solve the quadruple product system. This might lead to solving much larger routing problems in a speedy fashion.

The routing problems we solved were idealized problems where the solution could be predicted by simply studying the constraints. In reality however, this is not the case. We spent additional time investigating methods that might help speed up solving larger and more complex matrices that are characteristic of real-life circuit layout problems. Routing Problem 2 was self-contained and extensible; we therefore tried our code against a 750×300 matrix that resembled Routing Problem 2. Much to our surprise, the code worked without a single flaw and the anticipated solutions obtained. We were limited to trying much larger matrices by the computational power of our household laptops. We even went as far as to try our code against extremely complicated matrices such as the famed Hilbert matrix, generated using MATLAB's `hilb` and `gallery` functions.

Overall, every single issue that arose contributed to the refinement of the project. We learned when and where code optimization was possible and necessary, and how this can be achieved only through a continuous process of refinement and of course, a thorough background in the fundamentals of linear algebra. The procedures outlined mimic real-life scenarios where industry experts are constantly trying to optimize existing solutions, and where possible, find new ones. The project was a fulfilling experience in that it gave us a strong taste of how linear algebra can be exploited as a powerful tool in solving routine VLSI problems.

Appendices

A The MATLAB Code

```
% Interior Point Method Algorithm for solving the GLOBAL ROUTING problem
% Rajesh Swaminathan
% NE 112 - #20194189
%
% Created    : 10/24/2005
% Revised    : 10/31/2005
% Re-revised : 11/21/2005 (Routing Problem #1)
% Finalized  : 11/28/2005 (Routing Problem #2)

% Interior Point Method on the general (LP) Problem
%       Max c'x
%       such that Ax <= b

pfunction_new = c' * x;
k = 0;

while 1

    % STEP 1
    % present function value
    pfunction = pfunction_new;

    % STEP 2
    % Defining the distance of the current interior point from each of the
    % LENGTH(A) constraints
    vk = b - A*x;

    % STEP 3
    % Calculate the diagonal matrix
    Dk = diag(1./vk);

    % STEP 4
    % Finding the projection direction towards the optimal solution
    proj = A' * Dk * Dk * A;

    % Way 1 - Inverse (inefficient)
    % dx = inv(proj) * c;

    % Way 2 - mldivive (uses cholesky automatically)
    % dx = proj \ c;

    % Way 3 - Cholesky
    % dx = cholesky(proj, c);

    % Way 4 - Gauss-Seidel
    dx = gauss_seidel(proj, c);

    % Way 5 - Preconditioned Conjugate Gradient Method
    % [dx, flag] = pcg(proj, c, eps, 30);

    % STEP 5
    % Scale the projected direction dx
    dv = -A * dx;

    % STEP 6
    % Calculating the appropriate STEP LENGTH so that we stay in the
    % feasible region

    % Notes:
    %   1) vk is strictly postive
    %   2) all positive and zero values are discarded
    %   3) disvion by zero avoided

    for i=1:length(dv)
        if dv(i) < 0
            step(i) = vk(i) / dv(i);
        else
            step(i) = -1E30;
        end
    end

    % or,
    %
    % dv(dv >= 0) = -1E-30
    % step = vk ./ dv;

    alpha = gamma * max(step);

    % Calculate the new interior point
    x = x - alpha * dx;

    % STEP 7
    % Check the relative stopping criterion
    pfunction_new = c' * x;

    if abs((pfunction_new - pfunction) / (pfunction + eps)) < eps
        disp([num2str(k) ' iterations']);
        disp(x);
        disp('    ------');
        disp(pfunction_new);
        break
    end

    k = k + 1;

end
return;
```

B Global Routing Problems

Routing Problem 1

MAX $5y_1+y_2+5y_3+y_4+5y_5+y_6+5y_7+y_8+5y_9+y_{10}+5y_{11}+5y_{12}+y_{13}+5y_{14}+5y_{15}+y_{16}+5y_{17}+5y_{18}+5y_{19}$

$$\begin{aligned}
y_1+y_2 &\le 1\\
y_3+y_4 &\le 1\\
y_5+y_6 &\le 1\\
y_7+y_8 &\le 1\\
y_9+y_{10} &\le 1\\
y_{11} &\le 1\\
y_{12}+y_{13} &\le 1\\
y_{14} &\le 1\\
y_{15}+y_{16} &\le 1\\
y_{17} &\le 1\\
y_{18} &\le 1\\
y_{19} &\le 1
\end{aligned}$$

$$y_j \ge 0, \text{ for } j = 1, 2, \dots, 19$$

The result is summarized as follows:

```
10 iterations
    1.0000
    0.0000
    1.0000
    0.0000
    1.0000
    0.0000
    1.0000
    0.0000
    1.0000
    0.0000
    1.0000
    1.0000
    0.0000
    1.0000
    1.0000
    0.0000
    1.0000
    1.0000
    1.0000

    ------
   60.0000
```

```
Elapsed time: 0.601 s
```

Thus the optimal solution is

$y_1 = y_3 = y_5 = y_7 = y_9 = y_{11} = y_{12} = y_{14} = y_{15} = y_{17} = y_{18} = y_{19} = 1$; and
$y_2 = y_4 = y_6 = y_8 = y_{10} = y_{13} = y_{16} = 0$.

Routing Problem 2

$$\begin{aligned} \text{MAX} \quad & y_1 + 2y_2 + 3y_3 + 4y_4 + 5y_5 + 5y_6 + 4y_7 + 3y_8 + 2y_9 + y_{10} + \\ & 5y_{11} + 4y_{12} + 4y_{13} + 2y_{14} + 1y_{15} + 1y_{16} + 2y_{17} + 2y_{18} + 4y_{19} + 5y_{20} \end{aligned}$$

$$\begin{aligned} y_1 + y_{11} &\le 1 \\ y_2 + y_{12} &\le 1 \\ y_3 + y_{13} &\le 1 \\ y_4 + y_{14} &\le 1 \\ y_5 + y_{15} &\le 1 \\ y_6 + y_{16} &\le 1 \\ y_7 + y_{17} &\le 1 \\ y_8 + y_{18} &\le 1 \\ y_9 + y_{19} &\le 1 \\ y_{10} + y_{20} &\le 1 \end{aligned}$$

$$0 \le y_j \le 1, \text{ for } j = 1, 2, \ldots, 20$$

The result is summarized as follows:

```
11 iterations
    0.0000
    0.0000
    0.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    0.0000
    0.0000
    1.0000
    1.0000
    1.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    1.0000
    1.0000

    ------
   43.0000

Elapsed time: 0.581 s
```

Thus the optimal solution is

$y_4 = y_5 = y_6 = y_7 = y_8 = y_{11} = y_{12} = y_{13} = y_{19} = y_{20} = 1$; and
$y_1 = y_2 = y_3 = y_9 = y_{10} = y_{14} = y_{15} = y_{16} = y_{17} = y_{18} = 0$.

C Preconditioned Conjugate Gradient Method

The Preconditioned Conjugate Gradient Method (PCG) works by taking the system $\mathbf{Ax} = \mathbf{b}$ and placing it in quadratic form. For a positive definite matrix $\mathbf{A}$, the quadratic plane turns out to be parabolic. At the bottom of this plane, there exists a point where the derivative of the function equals zero, which is where our solution is. For a symmetric matrix $\mathbf{A}$, the derivative of any point will be equal to $\mathbf{Ax} - \mathbf{b}$. This can be derived using the quadratic equation and partial derivatives.

The steepest decent method takes an initial guess point on the plane and "slides down" into the paraboloid's bottom until it is close enough to the answer. This is achieved by creating a function that can be iterated through to obtain a solution:

$$\mathbf{P}'(\mathbf{x}) = \mathbf{Ax} - \mathbf{b}$$
$$\therefore -\mathbf{P}'(\mathbf{x}) = \mathbf{b} - \mathbf{Ax}$$

where $-\mathbf{P}'(\mathbf{x})$ will converge to zero as the number of iterations proceeds to infinity.

Since $-\mathbf{P}'(\mathbf{x})$ is equal to the distance between the evaluated guess and the constraints, we call it the residual. This relates to Step 2 of the interior point method.

$$\mathbf{R_i} = \mathbf{b} - \mathbf{Ax}_i$$

We also find that we can use the second part of Step 6 of the interior point method, which says:

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha\mathbf{dx}$$

We can think of $\mathbf{R}_0$ as the directional vector, ergo

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha\mathbf{R}_0$$

Also, we find that $-\mathbf{P}'(\mathbf{x}) \to 0$ as $\alpha \to 0$. Therefore taking the derivative of $\mathbf{P}(\mathbf{x}^{k+1})$ with respect to α yields $\mathbf{P}'(\mathbf{x}^{k+1})^T\mathbf{R}_0$ which equals zero at the end of convergence.

Since $-\mathbf{P}'(\mathbf{x}_0) = \mathbf{b} - \mathbf{Ax} = \mathbf{R}_0$, $\mathbf{P}'(\mathbf{x}_1)^T\mathbf{R}_0 = -\mathbf{R}_1^T\mathbf{R}_0 = 0$. Algebraic manipulation yields

$$\alpha = \mathbf{R}_0^T\mathbf{R}_0/\mathbf{R}_0^T\mathbf{A}\mathbf{R}_0$$

References

[1] I.Adler, N. Karmarkar, M.G.C. Resende, G.Veiga, “An implementation of Karmarkar’s algorithm for linear programming”, **Mathematical Programming**, Vol. 44, pp. 297-335, 1989.

[2] Dantzig G.B., Thapa M.N., “Linear Programming”, 1997.

[3] J. Shewchuk. “An introduction to the conjugate gradient method without the agonizing pain”, Technical Report CMUCS-TR-94-125, Carnegie Mellon University, 1994. (See also http://www.cs.cmu.edu/∼quake-papers/painless-conjugate-gradient.pdf.)