UNIVERSITY OF CALIFORNIA, SAN DIEGO

Power Network Verification and Optimization at Planning Stages

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science

by

Peng Du

Committee in charge:

Professor Chung-Kuan Cheng, Chair
Professor Fan Chung Graham
Professor Ronald Graham
Professor Jeffrey Remmel
Professor Jason Schweinsberg

2012

The dissertation of Peng Du is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____
Chair

University of California, San Diego

2012

DEDICATION

To my parents.

EPIGRAPH

Train yourself to let go
of everything you fear to lose.
—Master Yoda

TABLE OF CONTENTS

Signature Page iii

Dedication iv

Epigraph v

Table of Contents vi

List of Figures viii

List of Tables x

Acknowledgements xi

Vita xiii

Abstract of the Dissertation xv

Chapter 1 Introduction 1
1.1 Previous Works on Verification and Optimization 2
1.2 Contribution of our methods 4

Chapter 2 Power Network Verification 7
2.1 Single Current Source 9
2.1.1 Problem Formulation 9
2.1.2 Stage Partition 10
2.1.3 Worst-case Generation by Dynamic Programming 12
2.1.4 Dynamic Programming Speedup 12
2.1.5 Acceleration of the Verification Algorithm 18
2.1.6 Experimental Results and Summary 22
2.2 Multiple Current Sources 29
2.2.1 Problem Formulation 29
2.2.2 Hierarchical Current Constraints 30
2.2.3 Linear Programming on Submodular Polyhedron . 33
2.2.4 Method one: Dynamic Programming 38
2.2.5 Method two: Network Flow with Side Constraints 44
2.2.6 Method three: Submodular Flow 49
2.2.7 Experimental Results and Summary 51

Chapter 3 Power Grid Sizing 55
3.1 Problem Formulation 56
3.2 Voltage Drops versus Effective Resistances 57
3.3 Semidefinite Programming Reduction 58
3.4 Optimal Sizing by Convex Programming 59
3.4.1 Lanczos Algorithm for Objective Evaluation . . . 59
3.4.2 Derivative of Effective Resistances 61
3.5 Experimental Results on Regular Grids 62
3.6 Experimental Results on Practical Networks 64

Chapter 4 Optimization for dynamic power distribution network 69
4.1 Problem Formulation 70
4.2 The Overall Flow 72
4.3 Sensitivity Evaluation by Adjoint Network 73
4.4 Adaptive DFT Method for Circuit Simulation 76
4.4.1 Discrete Fourier Transform 78
4.4.2 Matrix Solver in Frequency Domain 78
4.4.3 Adaptive Flow for Frequency Partition 79
4.4.4 Time Complexity 80
4.5 Matrix Exponential Method for Circuit Simulation 81
4.5.1 Circuit Status Equation 81
4.5.2 Fast Explicit Numerical Integration Method . . . 84
4.5.3 Overall Numerical Integration Method 88
4.5.4 Stability and Error Analysis 90
4.5.5 Comparison with Euler Method 91
4.6 Nonlinear Programming Solver 93
4.7 Experimental Results 97

Chapter 5 Conclusion 103
5.1 Power Network Verification 104
5.2 Power Network Optimization 104

Bibliography 106

LIST OF FIGURES

Figure 1.1: A power distribution system from VRM to die. 2

Figure 2.1: The division of $[0, T]$ into $m = 5$ intervals. 10
Figure 2.2: The greedy input $G_j(k, i)$ in the time interval $[t_j, t_{j+1}]$. 11
Figure 2.3: The quadrangle inequality for the function W. 18
Figure 2.4: Lumped model of a power distribution system. 22
Figure 2.5: Impedance of the power distribution system. 24
Figure 2.6: Impulse response of the power distribution system. 24
Figure 2.7: Worst-case noise responses of the power distribution system at $t_r = 100ps$ and $t_r = 10ns$. 25
Figure 2.8: Worst-case load currents at $t_r = 100ps$ and $t_r = 10ns$. 25
Figure 2.9: Zoom-in part of the worst-case load currents at $t_r = 100ps$ and $t_r = 10ns$ around $50\mu s$. 26
Figure 2.10: Spectrum of the worst-case load currents at $t_r = 100ps$ and $t_r = 10ns$. 26
Figure 2.11: Worst-case voltage drop as a function of transition time. 27
Figure 2.12: Run time comparison of $O(n^2)$ and $O(n \log n)$ algorithm. 28
Figure 2.13: The flow network for the HCC in Table 2.2. 31
Figure 2.14: An example for the relationship between $\mathscr{D}$ and $P(\mathscr{D})$. 37
Figure 2.15: The division of $[0, T]$ into twelve intervals. 38
Figure 2.16: The greedy sum of load currents in the time interval $[t_j, t_{j+1}]$. . 39
Figure 2.17: The quadrangle inequality for the weight function. 42
Figure 2.18: The relation between L and Q. 43
Figure 2.19: Flow network construction. 46
Figure 2.20: The impulse responses of the power grid. 51
Figure 2.21: Zoom-in part of the impulse responses. 51
Figure 2.22: The worst-case currents with two different transition times. . . 52
Figure 2.23: The voltage noise waveform for $t_r = 5ps$. 53
Figure 2.24: The comparison of running time for different pairs of (n, s). . . 53

Figure 3.1: Optimization result for regular 2D power grids. 63
Figure 3.2: Effective resistances for a regular 10×10 grid. 63
Figure 3.3: A power grid model with four layers. 65
Figure 3.4: Sizing result for the practical power grid. 66
Figure 3.5: Voltage map of layer M1. 66
Figure 3.6: Sizing result when width range changes. 67
Figure 3.7: Voltage drops for different area distributions. 67

Figure 4.1: The overall flow of worst noise optimization. 72
Figure 4.2: The flow of predicting worst noise. 73
Figure 4.3: Sensitivity calculation relative to $\vec{I}(t)$. 74

Figure 4.4: Sensitivity calculation relative to $\vec{C}, \vec{R}$. 75
Figure 4.5: Compute output $v(t)$ in $[F_l, F_u]$ with period T, where $f_0 = 1/T$ and $F_l \leq kf_0 \leq F_u$. 76
Figure 4.6: Adaptive DFT flow for PDN simulation. 77
Figure 4.7: Example Circuit 83
Figure 4.8: Numerical Solution and Forward Euler 83
Figure 4.9: Numerical Solution and Backward Euler 83
Figure 4.10: Comparison between our method and forward Euler 92
Figure 4.11: Our method with different step sizes 92
Figure 4.12: A zoom-in view of figure 4.11 93
Figure 4.13: Results of Power Network Design 1 94
Figure 4.14: Power Network Model 97
Figure 4.15: Optimal resistances distribution. 99
Figure 4.16: Optimal capacitance distribution 100
Figure 4.17: The worst-case profiles of current sources. 101
Figure 4.18: Voltage profile at the origin. 102

LIST OF TABLES

Table 2.1: Circuit parameters of the power distribution system shown in Figure 2.4. 23
Table 2.2: An HCC example with seven load currents and twelve constraints 30
Table 2.3: The hierarchical current constraints. 52

Table 3.1: Comparison for the maximum effective resistance 64
Table 3.2: The network parameters of a power grid 65
Table 3.3: Comparison for different width ranges 67

Table 4.1: Runtime of Our Method and Backward Euler Method 94

ACKNOWLEDGEMENTS

Foremost, I would like to express my appreciation to my advisor Prof. Chung-Kuan Cheng for his support of my Ph.D study and research. Through my graduate period, he helped me in all the time of research with his patience, enthusiasm and a lot of ideas. He made great efforts to provide encouragement and give wise advice to me. I would not achieve any valuable result without him.

Chapter 2, in part, is a reprint of the material as it appears in International Symposium on Quality of Electronic Design(ISQED) 2010. Peng Du; Xiang Hu; Shih-Hung Weng; Amirali Arani; Xiaoming Chen; A. Ege Engin; Chung-Kuan Cheng, IEEE, 2010. The dissertation author was the primary investigator and author of this paper.

Chapter 3, in full, is a reprint of the material as it appears in IEEE International Conference on ASIC 2011. Peng Du; Shih-Hung Weng; Xiang Hu; Chung-Kuan Cheng, IEEE, 2011. The dissertation author was the primary investigator and author of this paper.

Chapter 4, in part, is currently being prepared for submission for publication of the material. Peng Du; Shih-Hung Weng; Xiang Hu; Chung-Kuan Cheng. The dissertation author was the primary investigator and author of this material. The section four, in part, is a reprint of the material as it appears in Asia and South Pacific Design Automation Conference(ASPDAC) 2010. Xiang Hu; Wenbo Zhao; Peng Du; Amirali Arani; Chung-Kuan Cheng, IEEE, 2010. The dissertation author was a cooperative author of this paper. The section five, in part, is a reprint of the material as it appears in IEEE Symposium on Circuits and Systems(ISCAS) 2011. Shih-Hung Weng; Peng Du; Cheng-Kuan Cheng, IEEE, 2011. The dissertation author was a cooperative author of this paper.

I would like to thank Prof. Fan Chung Graham and Ronald Graham who foster my interest in mathematics. Their vigor and wisdom for solving hard problems will encourage me to do a better job in my future life. I thank Prof. Jeff Remmel who shows me a lot of beautiful combinatorial results. His ideas on general theory of partitions with forbidden patterns and symmetric functions provide me with excellent examples on successful research. I thank Prof. Jason Schweinsberg

for his fruitful and intriguing probability course. His prudent and earnest attitude on mathematics always inspires me to learn and grow.

I am grateful to Prof. T. C. Hu for getting me involved in the interesting Blackjack project. I learned a lot from his kindness assistance on how we divide a large project into achievable steps. I also thank Prof. Andrew Kahng for his insightful comments on my projects.

My sincere thanks go to my colleagues Wenbo Zhao, Xiang Hu, and Shih-Hung Weng for discussing and sharing ideas with me. They help me get through difficult times and make my graduate study fun and exciting. I also wish to thank my colleagues Amirali Shayan, Xiaoming Chen, A. Ege Engin, my friends Kunfan Cheng, Jianjian Gao, Hao Zheng, He Zhu, Quan Chen, Yulei Zhang, Yuanzhe Wang, Guang Sun, Yijiang Shen and all the folks I have been worked with.

Last but not the least, I would like to thank my parents: Baocun Du and Zhixiong Zhao, for supporting me throughout my life.

VITA

1997-2001	B. S. in Computer Science, Zhejiang University, China
2001-2004	M. S. in Computer Science, Zhejiang University, China
2004-2005	Research Assistant in Information Engineering, The Chinese University of Hong Kong, China
2006-2009	M. A. in Applied Mathematics, University of California, San Diego, USA
2006-2012	Ph. D. in Computer Science, University of California, San Diego, USA

PUBLICATIONS

C. K. Cheng, P. Du, A. B. Kahng, S. H. Weng, "Low-Power Gated Bus Synthesis for 3D IC via Rectilinear Shortest-path Steiner Graph", *International Symposium on Physical Design(ISPD)*, 2012.

P. Du, W. Zhao, S. H. Weng, C. K. Cheng, R. L. Graham, "Character Design and Stamp Algorithms for Character Projection Electron-Beam Lithography", *Asia and South Pacific Design Automation Conference(ASPDAC)*, 2012.

C. K. Cheng, P. Du, A. B. Kahng, G. K. H. Pang, Y. Wang, N. Wong: "More realistic power grid verification based on hierarchical current and power constraints", *International Symposium on Physical Design(ISPD)*, 2011.

P. Du, S. H. Weng, X. Hu, C. K. Cheng, "Power Grid Sizing via Convex Programming", *IEEE International Conference on ASIC*, 2011.

S. H. Weng, P. Du, C. K. Cheng, "A Fast and Stable Explicit Integration Method by Matrix Exponential Operator for Large Scale Circuit Simulation", *IEEE Symposium on Circuits and Systems(ISCAS)*, 2011.

X. Hu, W. Zhao, P. Du, A. Arani, C. K. Cheng, "An adaptive parallel flow for power distribution network simulation using discrete Fourier transform", *Asia and South Pacific Design Automation Conference(ASPDAC)*, 2010.

X. Hu, P. Du, C. K. Cheng, "Exploring the Rogue Wave Phenomenon in 3D Power Distribution Networks", *IEEE Electrical Performance of Electronic Packaging(EPEP)*, 2010.

P. Du, X. Hu, S. H. Weng, A. Arani, X. Chen, A. E. Engin, C. K. Cheng, "Worst-case noise prediction with non-zero current transition times for early power distribution system verification", *International Symposium on Quality of Electronic Design(ISQED)*, 2010.

X. Hu, W. Zhao, P. Du, Y. Zhang, A. Arani, C. Pan, A. E. Engin, C. K. Cheng, "On the Bound of Time-Domain Power Supply Noise Based on Frequency-Domain Target Impedance", *IEEE System Level Interconnect Prediction(SLIP)*, 2009.

K. Zhou, P. Du, L. Wang, Y. Matsushita, J. Shi, B. Guo, H. Y. Shum, "Decorating Surfaces with Bidirectional Texture Functions", *IEEE Transactions on Visualization and Computer Graphics(TVCG)*, 2005.

ABSTRACT OF THE DISSERTATION

Power Network Verification and Optimization at Planning Stages

by

Peng Du

Doctor of Philosophy in Computer Science

University of California, San Diego, 2012

Professor Chung-Kuan Cheng, Chair

Power integrity has become a critical issue in nano-scale VLSI design. With technology scaling, the circuit integration density grows rapidly. However, the number of IO's dedicated for power does not scale up accordingly due to limited advancement in packaging technology. The increase of total current causes large voltage drops in the on-chip power grid, and the increase of clock frequencies results in large Ldi/dt noise due to the inductive effect of the power grid. Voltage drops degrade circuit timing performance while voltage bounces may cause reliability issues. On the other hand, the decrease in supply voltage leads to a smaller noise margin which makes the design of on-chip power grid an even more challenging task. As a result, full-chip power grid verification and optimization have become

essential for reliable chip design.

In the first part of this thesis, we propose novel methods of generating the worst-case noise for early power distribution system verification. These methods take into account the effect of the transition time of load currents, and thus allow a more realistic worst-case noise prediction. In the case of one current source, we introduce a dynamic programming algorithm on the time-domain impulse response of the power distribution system, and a modified Knuth-Yao Quadrangle Inequality Speedup is developed which reduces the time complexity of the algorithm to $O(nm \log n)$, where n is the number of discretized current values and m is the number of zeros of the system impulse response. In the case of multiple current sources, the dynamic programming algorithm is extended to generate the worst-case noise subject to a set of hierarchical current constraints. We show that the hierarchical constraints can be generalized into submodular polyhedron constraint and our algorithms still work. Furthermore, for other general magnitude and slope constraints of current sources, we propose the solutions by network flow and submodular flow which are more efficient than direct linear programming solution.

The second part of the thesis describes a power grid sizing method to minimize the worst voltage drop over all test locations and current source distributions. We reduce the original problem into a convex programming problem whose objective is to minimize the maximum effective resistance between the current entry node and all current exit nodes under the constraint of constant total wire area. In order to solve the convex programming problem efficiently, we adopt a Krylov space method to evaluate the effective resistances simultaneously and deduce a simple formula to update the derivative of effective resistance relative to perturbation of wire widths gradually. The proposed optimization method can also be applied to power grids in the real world, which are required to have small effective resistances among power stations for reducing the power losses during long distance transmission.

Finally, we propose a method for dynamic power distribution network verification and optimization at early design stages. This approach predicts and mini-

mizes the worst total voltage violation area for all outputs of a given on-chip power grid with multiple current sources. We assume duty-cycle constraints, group magnitude constraints and transition time constraints on current sources which makes the prediction more realistic. Simultaneously, we minimize the worst violation area through the allocation of decoupling capacitors (Decap) and controlled equivalent series resistors (ESR). Based on the simulation of adjoint network, the sensitivity of the violation area relative to current sources, Decap and ESR can be derived and the sequential quadratic programming method is adopted for optimization.

Chapter 1

Introduction

A *power distribution system* PDS (or *power distribution network* (PDN)) includes the voltage regulator module (VRM), motherboard, package and on-chip power grid (shown in Figure 1.1). The voltage regulator converts the system power supply to a voltage V_{dd} used for the integrated circuits. The board and package power planes connect the voltage regulator to the on-chip power grids, which supply the voltage to load circuits. Decoupling capacitors are placed hierarchically along the power distribution systems. Each hierarchy of the decoupling capacitors is effect within a certain range of frequencies to maintain low impedance for the PDS over the whole operation frequency. The PDS is typically modeled by RLC components or electromagnetic (EM) models, such as S-parameters. For early design-stage analysis, a lumped RLC circuit model can accurately describe the characteristics of a power distribution system [PMF08]. In this work, we first focus on the lumped PDS model and extend the proposed approaches to distributed models with multiple current sources afterwards.

In this chapter, we discuss about the motivation and previous works on power network verification and optimization in Section 1.1. Then we introduce the main framework of our methods in Section 1.2. The contribution and improvement of our work over previous methods are also described in Section 1.2.

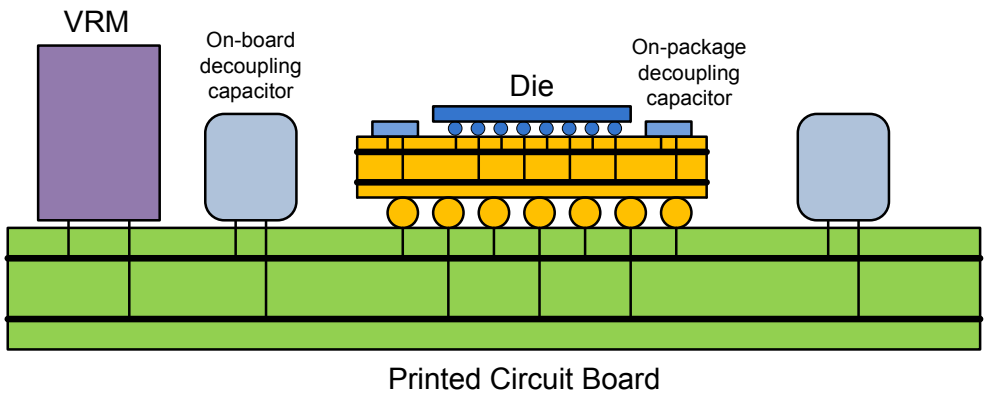


Figure 1.1: A power distribution system from VRM to die.

1.1 Previous Works on Verification and Optimization

With increasing clock frequency and decreasing supply voltage, power integrity becomes a critical issue. Due to the increasing number of signal I/O's which swallow the routing sources used for power delivery, design of off-chip power distribution networks is getting more and more challenging as well as on-chip power grid design. Traditional design methodologies focused on the frequency-domain behavior of power distribution systems and a lot of efforts have been made to minimize the PDS input impedance. However, the frequency-domain impedance does not always reflect the time-domain PDS noise [HZZ+09]. As a result, time-domain verification of power distribution systems has become essential to reliable design.

A typical way of performing PDS verification is based on simulation. For such an approach, one needs to know all the possible current waveforms drawn by the load circuits. However, PDS verification must be performed at the early-design stage, when full knowledge of the load currents is hardly available. To address this problem, efforts of finding the worst-case noise of a PDS without complete current information have been taken. A vectorless approach was introduced in [KN03] and extended in its following works [GN09, FNK07]. This approach gives an upper bound of the worst-case noise based on the current constraints without any simulation. In [HZZ+09], the worst-case noise was predicted based on the

ideal PDS step response. However, all those works assumed that the current has a zero transition time. In [FCN08], the worst-case current was generated by the accumulated step responses. Although it considered the maximum change between time units as a bound, the transition time within each time unit is still assumed to be zero. Such an assumption makes the worst-case current unrealizable, and leads to an overly pessimistic prediction for the worst-case noise, as will be shown later.

Power grid sizing is one of effective ways to optimize power network design. Many previous works [WC02][SVGY01][GK06][WMS05] have made a lot of effort on power grid sizing. Wang and Chen [WC02] proposed a sequential network simplex algorithm to size the widths of metal layers in a power network in order to minimize the total area with the IR drop constraint satisfied. In [SVGY01], Boyd et al. regarded the current demand as a random distribution and devise a heuristic method to determine the optimal wire widths for a power network subject to limits on voltage drops, total wire area, etc. Gupta and Kahng [GK06] derived a close form solution of IR drop based on the bull's eye phenomenon in the mesh-structured power grid. Wang and Marek-Sadowska [WMS05] proposed a multi-grid method to optimize power grids and decoupling capacitances simultaneously. Recently, convex programming methods play an important role on solving circuit optimization problems [VBG98][JC] because of its stability and efficiency. Vandenberghe et al. [VBG98] measure the signal propagation delay through the circuit as the dominant time constant which is a quasi-convex function of conductances and capacitances. Johnson and Chertkov [JC] optimized the power dissipation within the transmission network subject to the constraint on the cost of building the network. They have successfully reduced this problem into solving a sequence of convex programming problems.

With technology scaling, the power network is experiencing unprecedented noise, which causes significant delay variation of devices, or even logic failure. Recently, adding decoupling capacitors (Decap) between the power network and the ground becomes an effective and widely adopted approach to reduce the power network noise. In order to refine the Decap values, adjoint network methods have been applied to calculate the sensitivity of the violation area for circuit node with respect

to Decap change [SGS03][SSN03]. The sensitivity is then used as a sub-module in the nonlinear optimization solver. Because of the large number of simulations required for each iteration step, the computational complexity could be very high. To reduce the computational complexity, the merged adjoint network method was introduced and applied to calculate the sensitivity of the overall violation area with respective to Decap [LQT+06][FLH+04]. The idea is based on the superposition principle of linear circuits.

1.2 Contribution of our methods

In the first part of Chapter 2, we propose a novel method of predicting the worst-case noise of a power distribution system with single current source. This method takes into account the effect of the transition time of load currents, and thus allows a more realistic worst-case noise prediction. A dynamic programming algorithm is introduced on the time-domain impulse response of the power distribution system, and a modified Knuth-Yao Quadrangle Inequality Speedup is developed which reduces the time complexity of the algorithm to $O(nm \log n)$, where n is the number of discretized current values and m is the number of zeros of the system impulse response. With the algorithm, the worst-case noise behavior of the power distribution system is investigated with respect to the transition time. Experimental results show that assuming a zero current transition time leads to an overly pessimistic worst-case noise prediction.

In the second part of Chapter 2, we consider the power distribution network verification problem with multiple current sources. The proposed approach takes into account nonzero transition times of load currents and handles a set of hierarchical constraints for current magnitudes. The hierarchical current constraints not only give upper bounds on individual current sources, but also specify upper bounds on the total currents of separate groups of circuit blocks. We extend the dynamic programming algorithm adopted for single current source case to generate the worst-case currents. Then the algorithm is accelerated by a similar Knuth-Yao Quadrangle Inequality method. Moreover, we relax the hierarchical constraints

into submodular polyhedron constraints and our algorithms still work. If more general constraints are assumed for magnitude and slope of current sources, we introduce two algorithms via network flow and submodular flow to predict the worst case noise which have more time complexity than dynamic programming, but are much faster than the naive linear programming solution.

In Chapter 3, we devise a power grid sizing method to optimize the worst possible IR drop on an on-chip power network without explicit deriving the current distribution. We utilize the total current density/magnitude given in the design specification as the constraint for optimization of power grids. The optimization with only the total current density/magnitude constraint enables the early stage power grid verification and also avoids the effort of logic simulation for measuring average density/magnitude of every current source. Our method reduces the sizing problem into an effective resistance optimization problem which can be solved by convex programming efficiently. Experimental results show that our method can achieve up to 40% improvement for regular 2D grids and 7.32% improvement for a four layers power grid with only top two layers tunable over uniform sizing. Furthermore, the proposed optimization method can be applied not only to on-chip power grids, but also to power grids in the real world, which also need to reduce effective resistances among power stations for decreasing power losses during long distance transmission.

In Chapter 4, we predict and minimize the worst total voltage violation area for all outputs with multiple current stimuli. The proposed approach takes into account duty-cycle constraints, group magnitude constraints and transition time constraints on current sources which make the verification more realistic. The developed algorithm is based on the adjoint network approach and is thus not limited by the power grid models (e.g, RC, RLC, regular, irregular). By alternatively shifting between verification and optimization, we minimize the worst violation area through the allocation of decoupling capacitors (Decap) and controlled equivalent series resistors (ESR). In order to simulate the original network and adjoint network efficiently, we adopt two methods including adaptive DFT and exponential method which are much faster than standard SPICE library. Finally, we use the

sequential quadratic programming method to solve our non-linear programming problem.

Chapter 2

Power Network Verification

The goal of this chapter is to provide a practical solution for package and board designers to perform a verification of the power distribution system without knowing the exact on-chip load current information. Traditional power distribution network (PDN) analysis methods emphasize the frequency domain, and a great deal of work has been done to minimize PDN impedance. However, small PDN impedance does not always lead to small noise [HZZ+09]. Thus, power grid verification needs to be performed directly in the time domain.

Today, time-domain verification is usually based on exhaustive simulation. For such an approach, one needs to know all the possible current waveforms drawn by the circuits. Then power grid verification becomes a problem of finding the worst-case noise over all possible load currents. However, the full knowledge of current profiles drawn by the load circuits is usually not available before the completion of chip design. Even if the chip design is fixed, it is difficult to determine all the possible input patterns for the circuit and obtain the corresponding output currents. Furthermore, one needs to simulate the power distribution system with all the identified current waveforms for a full robustness check, which is prohibitively expensive.

The concept of "current constraints" was introduced in [KN03] to capture the uncertainty of the load currents and overcome the limitations of the simulation-based approach. This concept is reasonable because chip designers always have some knowledge about the circuit currents even though they may not know the

complete information. This knowledge may be based on the information from previous designs or from the system-level simulation at early design stage. Package and board designers can use such information for the early verification of the power distribution system. With this concept, the early power grid verification problem can be formulated as computing the worst-case noise under current constraints.

In [KN03] and its follow-up work [GN09, FNK07], a vectorless approach was proposed which formulated the verification problem as a set of linear programming instances and gave an upper bound of the worst-case noise subject to current constraints. However, only the current magnitudes were considered as the constraints, while the current transition times were assumed to be zero. In [FCN08], the currents were described using wavelets and the worst-case current was generated by accumulated step responses. Although the maximum current variations between time units were considered as constraints, the transition times of the current changes were still assumed to be zero. Based on our experiments, the assumption of zero transition time will lead to overly pessimistic worst-case noise. As a result, an additional current constraint on the minimum transition time is incorporated in our work to find the worst-case noise of the power distribution system.

We divide the power grid verification problem into two cases in 2.1 and 2.2 respectively. First, a single current source is assumed and we devise a dynamic programming algorithm to construct a current profile which generates the worst-case voltage. Both slope constraint and magnitude constraint on the current source are considered. We also improve the efficiency of our algorithm by a modified Knuth-Yao inequality. Second, we extend the verification problem into multiple current sources and a special class of magnitude constraint, called hierarchical constraints, is assumed. A greedy algorithm of linear programming under hierarchical constraints is proposed and further extended into submodular polyhedron. Then we modify the dynamic programming for single current source and generate worst-case output for multiple current sources with hierarchical constraints and total slope constraint. If current sources satisfy more general constraints on magnitude and slope, we develop two other strategies based on network flow and submodular flow

to construct worst current profiles.

2.1 Single Current Source

2.1.1 Problem Formulation

The aim is to identify the worst-case noise under the current constraints. One of the constraints on the load current regards the amplitude, which can be expressed as:

$$0 \leq i(t) \leq b, \ \ \forall t \geq 0, \tag{2.1}$$

where b represents the upper bound of the transient current $i(t)$. Under such a constraint, the current transition time, denoted by t_r, is defined as the time that the current takes to finish a transition from 0 to b or vice versa. We add another constraint that the transition time of the current has a lower bound of t_b, i.e., $t_r \geq t_b$. The constraint on the minimum transition time can be expressed in the form of the constraint on the maximum current slope as:

$$-\frac{b}{t_b} \leq \frac{di(t)}{dt} \leq \frac{b}{t_b}, \ \ \forall t \geq 0, \tag{2.2}$$

where b/t_b is the maximum absolute value of the current slope.

We denote the noise of the power distribution system by $v(t)$. For noise analysis, the voltage source V_{dd} is considered as zero. Thus, $v(t)$ can be calculated by convoluting the input current with the system impulse response as:

$$v(t) = \int_0^t z(\tau) i(t-\tau)\, d\tau, \tag{2.3}$$

where $z(\tau)$ is the impulse response of the power distribution system. Since the convolution operation has an accumulative effect on $v(t)$, we set the integration time, which is denoted by T, to be such that the impulse response has died down to some negligible value. By replacing the current $i(t-\tau)$ in (2.3) with a general function $f(\tau)$ and changing the notation τ to t, we can formulate the problem of finding the worst-case noise as:

$$max \qquad v(T) = \textstyle\int_0^T z(t) f(t)\, dt$$

$$s.t. \quad 0 \le f(t) \le b, \ \ \forall t \ge 0,$$
$$-C \le \tfrac{df(t)}{dt} \le C, \ \ \forall t \ge 0, \tag{2.4}$$

where $C = b/t_b$ is the upper bound of the slope of $f(t)$. Once $f(t)$ is identified, the corresponding worst-case load current can be generated by letting $i(t) = f(T - t)$.

Notice that the problem formulation in (4.17) calculates the worst-case noise for the maximum voltage drop according to the current direction shown in Figure 2.4. The inductive effect of power distribution system may cause Ldi/dt noise, which includes both voltage drop and overshoot. For maximum voltage overshoot, the problem can be formulated simply by minimizing the object function in (4.17) under the same constraints. In the following of this section, we focus on generating the worst-case noise for the maximum voltage drop only.

2.1.2 Stage Partition

To solve the optimization problem (2.3), we first divide the time range $[0, T]$ into m intervals $[t_0 = 0, t_1], [t_1, t_2], \ldots, [t_{m-1}, t_m = T]$ such that $z(t)$ is either positive or negative in each interval as shown in Figure 2.15. We also choose $n + 1$ sample points $u_0 = 0, u_1, \ldots, u_{n-1}, u_n = b$ in the range of the current $[0, b]$ where n depends on the precision requirement.

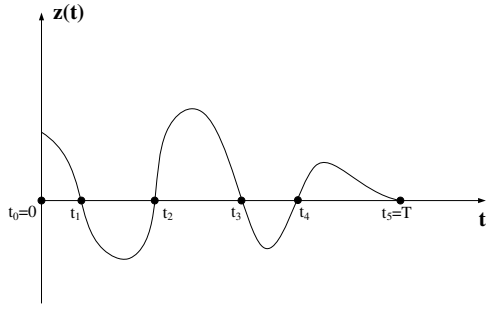
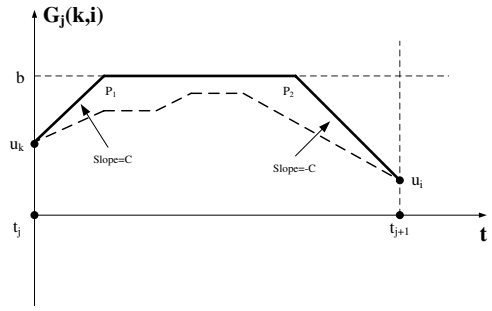
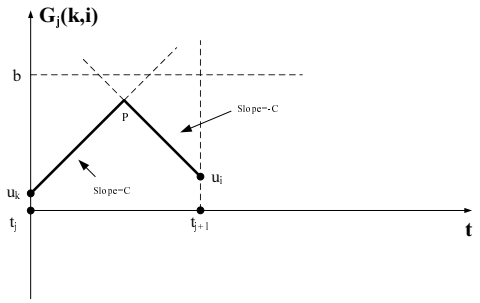


Figure 2.1: The division of $[0, T]$ into $m = 5$ intervals.

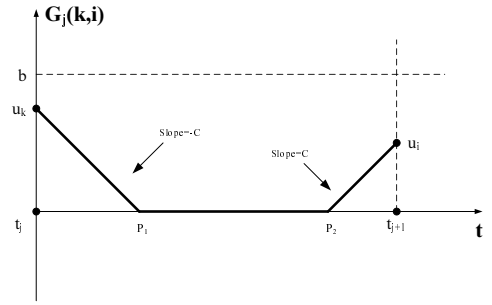
Given a time interval $[t_j, t_{j+1}]$, we define $G_j(k, i)$ to be the worst-case $f(t)$ starting with u_k at time t_j and ending with u_i at time t_{j+1}. Let $V_j(k, i)$ be the corresponding worst-case noise and we have $V_j(k, i) = \int_{t_j}^{t_{j+1}} z(t)(G_j(k, i)(t))\, dt$. We can compute $G_j(k, i)$ easily by a greedy algorithm. As shown in Figure 2.2(a) and



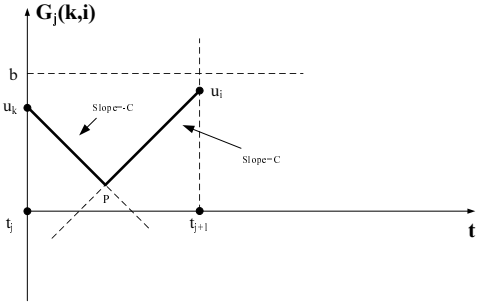
(a) $z(t)$ is positive between t_j and t_{j+1}.



(b) $z(t)$ is positive between t_j and t_{j+1}.



(c) $z(t)$ is negative between t_j and t_{j+1}.



(d) $z(t)$ is negative between t_j and t_{j+1}.

Figure 2.2: The greedy input $G_j(k,i)$ in the time interval $[t_j, t_{j+1}]$.

2.2(b), if $z(t)$ is positive in $[t_j, t_{j+1}]$, we draw a line starting from u_k with slope C and another line ending at u_i with slope $-C$. If these two lines intersect at a point P below the upper bound b of the current, $G_j(k,i)$ should be $u_k \to P \to u_i$ as in Figure 2.2(b). Otherwise, let P_1 and P_2 be the intersection points between these two lines and the upper bound b respectively. We take $u_k \to P_1 \to P_2 \to u_i$ as in 2.2(a) to be $G_j(k,i)$. If $z(t)$ is negative in $[t_j, t_{j+1}]$, $G_j(k,i)$ can be constructed similarly as shown in Figure 2.2(c) and 2.2(d). Without loss of generality, we use the case in Figure 2.2(a) to prove the correctness of the greedy algorithm. Let $G'_j(k,i)$ be any non-greedy $f(t)$ as the dotted curve in Figure 2.2(a), then we have $G'_j(k,i) \leq G_j(k,i)$ in the interval $[t_j, t_{j+1}]$. Since $z(t)$ is positive in $[t_j, t_{j+1}]$, we have $\int_{t_j}^{t_{j+1}} z(t)(G'_j(k,i)(t)) \leq V_j(k,i)$, which shows $G_j(k,i)$ is the worst-case $f(t)$ with certain constraints showed before.

2.1.3 Worst-case Generation by Dynamic Programming

Let $OPT(j,i)$ with $j \in [0,m]$ and $i \in [0,n]$ be the worst-case noise generated by the currents stopping at time t_j with value u_i. The base case and recursive formula for the function OPT should be

$$\begin{aligned} OPT(0,i) &= 0 \quad \text{for all } i \in [0,n] \\ OPT(j+1,i) &= \max_{0 \le k \le n} (OPT(j,k) + V_j(k,i)) \end{aligned} \tag{2.5}$$

which gives us a dynamic programming algorithm to compute all values of the function OPT and the maximum $v(T)$ is the largest one in the set $\{OPT(m,i) : i \in [0,n]\}$. Notice that the dynamic programming algorithm only considers the time stages $0, t_1, t_2, \ldots, t_{m-1}, T$, which reduces the time complexity of the computation significantly. We also have the following theorem for the worst-case $f(t)$.

Theorem 1. *Suppose $f_m(t)$ is the worst-case $f(t)$ which maximizes $v(T)$, we have $\frac{df_m(t)}{dt} = 0, C$ or $-C$.*

Proof. $f_m(t)$ is constructed by a bunch of greedy $G_j(k,i)$ through the OPT function. The slope of each $G_j(k,i)$ can only take values $0, C, -C$ as shown in Figure 2.16. So $f_m(t)$ has the same property, i.e. $\frac{df_m(t)}{dt} = 0, C or -C$. □

2.1.4 Dynamic Programming Speedup

In this section, we survey several ideas proposed in [Yao80][EGG88] which leads to our acceleration algorithm for power network verification. The basic strategy for dynamic programming speedup is to reduce the redundancy in the number of total states, the number of states which a specific state depends on and the time complexity of each state transition. The paper [Yao80] considers the problem of optimal binary search tree and proposes a general speedup technique via quadrangle inequality on weight function. Eppstein et al. [EGG88] extends the idea to many other types of dynamic programming problems with monotonic weight functions.

We first consider a dynamic programming problem with the following recursive formula:

$$\begin{aligned} d(i,i) &= 0, \\ d(i,j) &= \min_{i<k\leq j}\{d(i,k-1)+d(k,j)\}+w(i,j), \end{aligned} \tag{2.6}$$

where $1 \leq i, j \leq n$ and $w(i,j)$ is a giving weight function satisfying the quadrangle inequality:

$$w(i_1,j_1)+w(i_2,j_2) \leq w(i_1,j_2)+w(i_2,j_1) \tag{2.7}$$

for any $i_1 \leq i_2 \leq j_1 \leq j_2$. A naive method takes $O(n^3)$ time to evaluate the function $d(i,j)$ which can be reduced into $O(n^2)$ via decreasing the number of states considered in transition.

Lemma 1. *The function $d(i,j)$ also satisfies the quadrangle inequality, i.e.*

$$d(i_1,j_1)+d(i_2,j_2) \leq d(i_1,j_2)+d(i_2,j_1)$$

for any $i_1 \leq i_2 \leq j_1 \leq j_2$.

Proof. We prove the lemma by induction on $l = j_2 - i_1$. If $l = 1$, then either $i_1 = i_2$ or $j_1 = j_2$ and the quadrangle inequality holds trivially. Now we show the inequality by dividing the values of i_1, i_2, j_1, j_2 into the following cases:

- $i_1 < i_2 = j_1 < j_2$: Let k be an index satisfying $d(i_1,j_2) = d(i_1,k-1) + d(k,j_2) + w(i_1,j_2)$. Without loss of generality, we assume $k \leq j_1$. Then we have

$$\begin{aligned} & d(i_1,j_1)+d(i_2,j_2) \\ = \;& d(i_1,j_1)+d(j_1,j_2) \\ \leq \;& w(i_1,j_1)+d(i_1,k-1)+d(k,j_1)+d(j_1,j_2) \\ \leq \;& w(i_1,j_2)+d(i_1,k-1)+d(k,j_1)+d(j_1,j_2) \\ \leq \;& w(i_1,j_2)+d(i_1,k-1)+d(k,j_2) \\ = \;& d(i_1,j_2) \\ = \;& d(i_1,j_2)+d(i_2,j_1) \end{aligned}$$

- $i_1 < i_2 < j_1 < j_2$: Let k_1 be an index satisfying $d(i_1, j_2) = d(i_1, k_1 - 1) + d(k_1, j_2) + w(i_1, j_2)$ and k_2 be an index satisfying $d(i_2, j_1) = d(i_2, k_2 - 1) + d(k_2, j_1) + w(i_2, j_1)$. Without loss of generality, we assume $k_1 \leq k_2$ which implies $i_1 < k_1 \leq k_2 \leq j_1$. Then we have

$$\begin{aligned}
& d(i_1, j_1) + d(i_2, j_2) \\
\leq\ & (d(i_1, k_1 - 1) + d(k_1, j_1) + w(i_1, j_1)) + (d(i_2, k_2 - 1) + \\
& d(k_2, j_2) + w(i_2, j_2)) \\
\leq\ & (w(i_1, j_2) + w(i_2, j_1)) + (d(i_1, k_1 - 1) + d(i_2, k_2 - 1)) + \\
& (d(k_1, j_1) + d(k_2, j_2)) \\
\leq\ & (w(i_1, j_2) + w(i_2, j_1)) + (d(i_1, k_1 - 1) + d(i_2, k_2 - 1)) + \\
& (d(k_1, j_2) + d(k_2, j_1)) \\
=\ & (d(i_1, k_1 - 1) + d(k_1, j_2) + w(i_1, j_2)) + (d(i_2, k_2 - 1) + \\
& d(k_2, j_1) + w(i_2, j_1)) \\
=\ & d(i_1, j_2) + d(i_2, j_1)
\end{aligned}$$

□

We define $d_k(i, j)$ to be the k^{th} candidate of $d(i, j)$ and $s(i, j)$ to be the best candidate index which generates $d(i, j)$, i.e.

$$d_k(i, j) = d(i, k - 1) + d(k, j) + w(i, j)$$

$$s(i, j) = \max\{k | d(i, j) = d_k(i, j)\}$$

Lemma 2. *The function $s(i, j)$ is monotonic in both dimensions, i.e.*

$$s(i, j) \leq s(i, j + 1) \leq s(i + 1, j + 1)$$

Proof. By symmetry, we only show the inequality $s(i, j) \leq s(i, j + 1)$ holds. If $i = j$, the inequality is trivial. Let k_1, k_2 be two indexes satisfying $i < k_1 \leq k_2 \leq j$

and $d_{k_2}(i,j) \leq d_{k_1}(i,j)$, we have

$$\begin{aligned}
& d_{k_2}(i,j+1) \\
= \;& d(i,k_2-1) + d(k_2,j+1) + w(i,j+1) \\
= \;& (d(i,k_2-1) + d(k_2,j) + w(i,j)) + (d(k_2,j+1) - d(k_2,j)) + \\
& (w(i,j+1) - w(i,j)) \\
= \;& d_{k_2}(i,j) + (d(k_2,j+1) - d(k_2,j)) + (w(i,j+1) - w(i,j)) \\
\leq \;& d_{k_1}(i,j) + (d(k_2,j+1) - d(k_2,j)) + (w(i,j+1) - w(i,j)) \\
= \;& d(i,k_1-1) + (d(k_1,j) + d(k_2,j+1) - d(k_2,j)) + w(i,j+1) \\
\leq \;& d(i,k_1-1) + d(k_1,j+1) + w(i,j+1) \\
= \;& d_{k_1}(i,j+1)
\end{aligned}$$

which means that if k_2 is a larger and better candidate than k_1 for $d(i,j)$, then it will be a better candidate than k_1 for $d(i,j+1)$. Therefore, based on the definition of the function s, we conclude that $s(i,j) \leq s(i,j+1)$. □

Now we can achieve an optimized recursive formula for the function $d(i,j)$ as follows:

$$\begin{aligned}
d(i,i) &= 0 \\
d(i,j) &= \min_{s(i-1,j) \leq k \leq s(i,j+1)} \{d(i,k-1) + d(k,j)\} + w(i,j)
\end{aligned}$$

We evaluate the two dimensional array $d(i,j)$ corresponding to the increasing order of the value $j-i$, i.e. from the main diagonal line nearest to the origin, to the main diagonal line farthest to the origin, by taking (i,j) as coordinates in a 2D plane. Since the total cost for computing $d(i,j)$ in a diagonal line is $O(n)$, we can obtain all values of $d(i,j)$ in $O(n^2)$ time.

The formulation 2.6 shows up in many practical applications. The original one given in [Yao80] is the *optimal binary search tree* problem. In an instance of *optimal binary search tree* problem, n keys $a_1 \leq a_2 \leq \cdots \leq a_n$ are given and for each key a_i, there is a weight c_i. The objective is to organize the keys into a binary

search tree so that the sum of $c_i l_i$ for all i is minimal where l_i is the depth of a_i in the tree. If we define $d(i,j)$ to be the cost of the best binary search tree to organize the keys from a_i to a_j, we can write the recursive formula for $d(i,j)$ as follows:

$$d(i,j) = \min_{i<k<j}\{d(i,k-1) + d(k+1,j)\} + w(i,j),$$

where $w(i,j) = c_i + c_{i+1} + \cdots + c_j$. Clearly, $w(i,j)$ satisfies quadrangle inequality since $w(i_1,j_1)+w(i_2,j_2) = w(i_1,j_2)+w(i_2,j_1)$ for any $i_1 \le i_2 \le j_1 \le j_2$. Therefore, $d(i,j)$ can be evaluated in $O(n^2)$ time rather than $O(n^3)$ by naive computing.

Eppstein et al. [EGG88] considers the one dimensional recursive formula shown below:

$$\begin{aligned} f(1) &= 0, \\ f(j) &= \min_{1\le i<j}\{f(i) + w(i,j)\}, \quad 1 < j \le n, \end{aligned} \tag{2.8}$$

where $w(i,j)$ satisfies the quadrangle inequality. We define $g(i,j) = f(i) + w(i,j)$ which is the i^{th} candidate for evaluating $f(j)$. Then $g(i,j)$ also satisfies the quadrangle inequality and we can prove the following monotonic property of $g(i,j)$:

Lemma 3. *If $1 \le i_1 < i_2 \le n$ and $g(i_2,j) \le g(i_1,j)$, then for any $k > j$, we have $g(i_2,k) \le g(i_1,k)$.*

Proof.

$$\begin{aligned} & g(i_2,k) \\ = \;& f(i_2) + w(i_2,k) \\ = \;& (f(i_2) + w(i_2,j)) + (w(i_2,k) - w(i_2,j)) \\ = \;& g(i_2,j) + (w(i_2,k) - w(i_2,j)) \\ \le \;& g(i_1,j) + (w(i_2,k) - w(i_2,j)) \\ = \;& f(i_1) + (w(i_1,j) + w(i_2,k)) - w(i_2,j) \\ \le \;& f(i_1) + (w(i_2,j) + w(i_1,k)) - w(i_2,j) \\ = \;& f(i_1) + w(i_1,k) \\ = \;& g(i_1,k) \end{aligned}$$

□

The lemma means that if i_2 is a better candidate index than i_1 for $f(j)$, then it will be a better candidate index than i_1 for any $f(k)$ with $k > j$. We define $i_j^* = \max\{i | f(j) = f(i) + w(i, j)\}$ to be the largest best candidate index for $f(j)$ and we have $i_1^* \leq i_2^* \leq \cdots \leq i_n^*$ based on the lemma. We define $p(i_1, i_2)$ for $i_1 < i_2$ to be the first j so that i_2 becomes a better candidate index than i_1 for $f(j)$. During the computation process of $f(j)$, we maintain a candidate index dequeue L and a dequeue S such that L contains current available candidate indexes sorting in increasing order and S contains the values of $p(i_1, i_2)$ for every adjacent pairs i_1, i_2 in L. L and S are initialized as empty and we obtain the values of $f(j)$ in the increasing order of j. When evaluating $f(j_0)$ for a specific index j_0, we first remove the head elements of L and S as long as the head of S is equal to j_0. Then $i_{j_0}^*$ will be the first index in L. The second step is to remove the tail elements of L and S as long as $p(i_t, j_0)$, where i_t is the tail element of L, is less than or equal to the tail element of S. Since each candidate index is inserted into and removed from the dequeue L at most once, the total time complexity for evaluating $f(j)$ is $O(nT)$ where T is the unit time of computing the function $p(i_1, i_2)$. Generally, $p(i_1, i_2)$ can be found by binary search and T is bounded by $O(\log n)$. In some special cases, we can obtain $p(i_1, i_2)$ in constant time. Hence, the total running time is $O(n \log n)$ or $O(n)$ rather than $O(n^2)$ by evaluating the recursive formula naively.

Many scheduling problem can be solved by dynamic programming algorithms with form . For example, given n jobs $J_1, J_2, \ldots, J_n$ with processing time P_i and weight C_i for job J_i running on a single processor sequentially. We can divide the jobs into several blocks $B_1, B_2, \ldots, B_m$. The time $R(B_j)$ for processing a block B_j includes a processor initial time T, plus the total processing time for jobs in B_j. The finishing time $F(J_i)$ for a job J_i in B_j is equal to $\sum_{1 \leq k \leq j} R(B_k)$ and the cost for J_i is $C_i * F(J_i)$. The objective is to find a block partition so that the total cost is smallest. If J_i is the last job of a block B_j, we define $f(i)$ to be the smallest cost for jobs before J_i (including J_i), plus $\sum_{i < k \leq n} C_k * F(J_i)$. We can

derive the recursive formula of $f(i)$ as below:

$$f(i) = \min_{1 \le k < i} \{f(k) + w(k, i)\},$$

where $w(k,i) = (T + \sum_{k<r\le i}(P_r))(\sum_{k<r\le n}(C_r))$. We can show that the weight function w satisfies quadrangle inequality and the acceleration method described above applies.

2.1.5 Acceleration of the Verification Algorithm

If we compute all values of the OPT function directly using the recursive formula, the time complexity is $O(n^2 m)$. In this section, we will prove some useful properties of our problem which lead to an accelerated dynamic programming algorithm with only $O(nm \log n)$ time complexity.

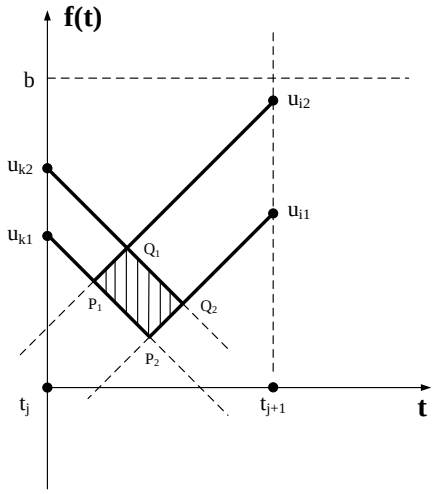


Figure 2.3: The quadrangle inequality for the function W.

Without loss of generality, we assume $z(t)$ is negative in the time interval $[t_j, t_{j+1}]$. Let $W_j(k,i)$ be the absolute value of $V_j(k,i)$ and $F_j(k,i)$ be the candidate corresponding to k for $OPT(j+1,i)$, i.e. $F_j(k,i) = OPT(j,k) - W_j(k,i)$. Some useful properties of the functions W and F are given as follows:

Lemma 4. *$W_j(k_2,i_2) - W_j(k_1,i_2) \le W_j(k_2,i_1) - W_j(k_1,i_1)$ for any $0 \le k_1 < k_2 \le n$ and $0 \le i_1 < i_2 \le n$.*

Proof. In Figure 2.17, $W_j(k_2, i_2)$ is the current of the area below $u_{k2} \to Q_1 \to u_{i2}$ convoluted with $|z(t)|$ and $W_j(k_1, i_2)$ is the current of the area below $u_{k1} \to P_1 \to u_{i2}$ convoluted with $|z(t)|$. Therefore, $W_j(k_2, i_2) - W_j(k_1, i_2)$ is the current of the quadrangle area u_{k2}, Q_1, P_1, u_{k1} convoluted with $|z(t)|$. Similarly, $W_j(k_2, i_1) - W_j(k_1, i_1)$ is the current of the quadrangle area u_{k2}, Q_2, P_2, u_{k1} convoluted with $|z(t)|$, which is larger than or equal to $W_j(k_2, i_2) - W_j(k_1, i_2)$ since u_{k2}, Q_2, P_2, u_{k1} contains u_{k2}, Q_1, P_1, u_{k1}. □

Note that lemma 8 is the quadrangle inequality as in [Yao80] for the function W. However, the speedup method using in [Yao80] is not suitable for our case since we can not deduce the quadrangle inequality for the function OPT from the one for W. Here we give a novel alternative of the Knuth-Yao speedup based on the following lemma and reduce the complexity of our algorithm from $O(n^2 m)$ to $O(nm \log n)$ successfully.

Lemma 5. *Suppose $k_1 < k_2$, $i_1 \in [0, n]$ and $F_j(k_1, i_1) \leq F_j(k_2, i_1)$, then for any $i_2 > i_1$, we have $F_j(k_1, i_2) \leq F_j(k_2, i_2)$.*

Proof.

$$
\begin{aligned}
& F_j(k_1, i_2) \\
= \;& OPT(j, k_1) - W_j(k_1, i_2) \\
= \;& (OPT(j, k_1) - W_j(k_1, i_1)) + (W_j(k_1, i_1) - W_j(k_1, i_2)) \\
= \;& F_j(k_1, i_1) + (W_j(k_1, i_1) - W_j(k_1, i_2)) \\
\leq \;& F_j(k_2, i_1) + (W_j(k_1, i_1) - W_j(k_1, i_2)) \text{ by the condition} \\
\leq \;& F_j(k_2, i_1) + (W_j(k_2, i_1) - W_j(k_2, i_2)) \text{ by lemma 8} \\
= \;& OPT(j, k_2) - W_j(k_2, i_2) \\
= \;& F_j(k_2, i_2)
\end{aligned}
$$

□

Based on lemma 9, suppose $k_1 < k_2$, we can find the smallest i_0 such that $F_j(k_1, i) \leq F_j(k_2, i)$ whenever $i \geq i_0$ within $O(\log n)$ time by the binary search. We define this function as $i_0 = GetTransPos(j, k_1, k_2)$.

Let S be the sequence of candidates k from small to large for the worst-case noise $OPT(j+1,i)$. Let Q be a priority queue which has three operations $GetMin, DeleteMin$ and Add. $Q.GetMin()$ and $Q.DeleteMin()$ will return and delete the minimum element of the queue respectively. $Q.Add(e)$ means inserting the element e to the queue. All these three operations can be finished in the time complexity of $O(\log n)$. Each element of Q contains three fields. The first field is the key for the priority and it indicates the position i_0 as we showed in the end of the last section. The last two fields are k_1, k_2 corresponding to i_0. Now our accelerated dynamic programming algorithm can be shown in the pseudocode "algorithm $GetOPT$", which runs in the time complexity of $O(n \log n)$ instead of $O(n^2)$ for the naive computation of the function OPT.

Algorithm 1: GetOPT

Input: $OPT(j,i)$ for all $i \in [0,n]$

Output: $OPT(j+1,i)$ for all $i \in [0,n]$

```
1  S={0,1,...,n};
2  Q={[GetTransPos(j,k1,k2),k1,k2] : k1,k2 are adjacent in S};
3  for i = 0,...,n do
4      [i0,k1,k2] = Q.GetMin();
5      while i0 == i do
6          Q.DeleteMin();
7          if k1,k2 are adjacent in S then
8              Delete k1 in S;
9              Find k3 next to k2 on the left in S;
10             Q.Add([GetTransPos(j,k3,k2),k3,k2]);
11         end
12         [i0,k1,k2] = Q.GetMin();
13     end
14     q=The first element in S;
15     OPT(j+1,i) = OPT(j,q) - W_j(q,i);
16 end
```

The correctness proof of the algorithm is given as follows. First we claim that for each iteration i, the sequence S contains all possible candidates k to compute $OPT(j+1,i)$ after step 13 in algorithm $GetOPT$. Initially, S contains all values from 0 to n as in step 1. For each adjacent pair k_1, k_2 in S, the priority queue Q stores a node with three fields indicating k_1,k_2 and the first position i_0 after which k_2 will be a better candidate, i.e. $F_j(k_2,i) \geq F_j(k_1,i)$ whenever $i \geq i_0$. In step 4, we retrieve the node $[i_0,k_1,k_2]$ from Q with minimum i_0. If i_0 is equal to i, we know that k_1 will not be a better candidate than k_2 for the present and future i. Therefore, it is reasonable to delete k_1 in S showed in step 8. After the deletion, the element k_3 next to k_1 on the left originally will be adjacent to k_2 in S now. To maintain the properties of Q, we add the new node for the adjacent pair k_3, k_2 as in step 10. It should be noticed that some adjacent pair will be unavailable anymore during the deletion. For example, if k_1 is deleted, the information for the pair k_3, k_1 becomes useless. If we come across such pairs in Q, we can detect them in step 7 and skip the operations from step 8 to 10. After the iteration from step 5 to 13, we delete all impossible candidates from now on and the claim that S contains all possible candidates k has been proved.

Based on the previous argument, we also see that after step 13, for any $k_1, k_2 \in S$ and $k_1 < k_2$, $F_j(k_1,i)$ is larger than $F_j(k_2,i)$ since otherwise k_1 should be deleted in or before the iteration i, i.e. $F_j(\cdot,i)$ monotonically decreases. Hence in steps 14 and 15, we can take the first element in S to compute $OPT(j+1,i)$ since it is the best candidate.

Theorem 2. *The algorithm GetOPT runs in the time complexity of* $O(n \log n)$.

Proof. We can implement the sequence S by a doubly-linked list and we also need a pointer array which gives the position of each k in S. Now step 8 and 9 only take constant time. Furthermore, we notice that each candidate k can only be deleted once. Hence the operations from step 6 to 12 will run at most n times in the whole process of algorithm $GetOPT$. Based on our knowledge that all three operations of the priority queue runs in $O(\log n)$ time and computing the function $GetTransPos$ also takes $O(\log n)$ time, we have shown that the total time complexity of algorithm $GetOPT$ is $O(n \log n)$. □

Theorem 3. *We can compute the OPT function within time* $O(nm \log n)$.

Proof. We only need to initiate $OPT(0, i) = 0$ for all $i \in [0, n]$ with respect to equation 2.17 and call the algorithm GetOPT m times for $j = 1, 2, \ldots, m$ increasingly. □

2.1.6 Experimental Results and Summary

In this section, the lumped power distribution system model as shown in Figure 2.4 is studied with the proposed approach. Without loss of generality, the worst-case noise behavior of the power supply is investigated and we only consider the power network as shown in Figure 2.4. V_{dd} is the external system power supply.

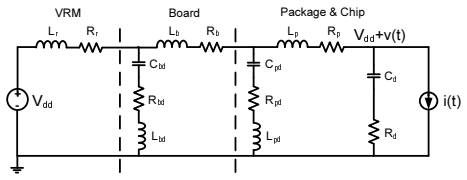


Figure 2.4: Lumped model of a power distribution system.

R_r and L_r represent the output impedance of the VRM and the impedance of the current path from the VRM to the on-board decoupling capacitors. R_b and L_b represent the impedance of the current path from board capacitors to the package, the impedance of the board-package interface, and the impedance of the package planes. R_p and L_p represent the impedance of the current path from on-package decoupling capacitors to die and the on-chip power grid. C_{bd} and C_{pd} are the on-board and on-package decoupling capacitors, respectively, while R_{bd}, L_{bd}, R_{pd} and L_{pd} represent the associated equivalent series resistance (ESR) and equivalent series inductance (ESL) of C_{bd} and C_{pd}, respectively. Similarly, C_d and R_d signify the effective on-chip decoupling capacitor and the associated ESR, respectively. Note that C_d includes the intrinsic capacitance of the non-switching transistors in a circuit and the dedicated decoupling capacitance [SPKF05]. The ESL of C_d is ignored since it is negligible for on-chip decoupling capacitors. The switching of the load circuit is represented by the current source $i(t)$. The impedance between

the on-chip decoupling capacitor and the load current is ignored assuming that the decoupling capacitor is placed sufficiently close to the load circuit [PSKF08].

Consider a microprocessor where we assume the peak value of the load current is $50A$. The values of the circuit components are listed in Table 2.1. Figure 2.5 plots the impedance of the power distribution system. There are three peaks at $19.8KHz$, $465KHz$ and $166MHz$, which correspond to the anti-resonance at VRM-board, board-package and package-die interface, respectively. The values of these peaks are $2.09m\Omega$, $1.69m\Omega$ and $3.23m\Omega$, respectively. Figure 2.6 displays the impulse response of the power distribution system up to $10ns$. Due to the anti-resonance peaks in the impedance, the impulse response oscillates with time. Notice that Figure 2.6 only shows the oscillation corresponding to the highest resonant frequency. The oscillations that correspond to middle and low resonant frequencies can be observed with longer time. For accurate worst-case noise calculation, the simulation time should be long enough until the impulse response dies down to a negligible value.

Table 2.1: Circuit parameters of the power distribution system shown in Figure 2.4.

Circuit parameter	Value	Circuit parameter	Value
R_r	$1m\Omega$	L_r	$10nH$
C_{bd}	$5mF$	R_{bd}	$0.5m\Omega$
L_{bd}	$0.3nH$	R_b	$0.3m\Omega$
L_b	$0.2nH$	C_{pd}	$250\mu F$
R_{pd}	$0.8m\Omega$	L_{pd}	$1pH$
R_p	$0.1m\Omega$	L_p	$1pH$
C_d	$500nF$	R_d	$0.5m\Omega$

Based on the impulse response, the developed algorithm is applied to generate the worst-case noise with two transition time: $t_r = 100ps$ and $t_r = 10ns$. The worst-case noise responses of these two cases are plotted in Figure 2.7. Note that the worst-case noise responses for the maximum voltage drop are generated here. At $t_r = 100ps$, the maximum voltage drop is $183.3mV$, and is $124.3mV$ when

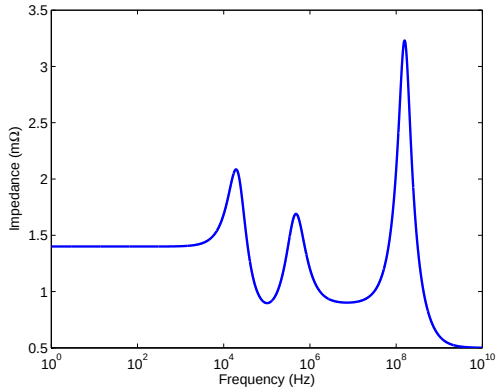


Figure 2.5: Impedance of the power distribution system.

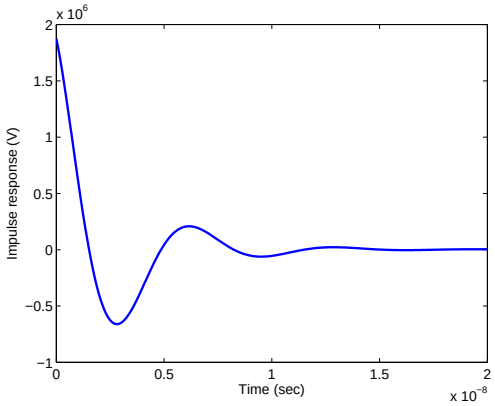


Figure 2.6: Impulse response of the power distribution system.

$t_r = 10ns$. Figure 2.8 and Figure 2.10 compare the time-domain worst-case currents and their spectrum, respectively. The worst-case currents typically follow the oscillation pattern of the PDS impulse response. However, at large transition time, the transition of the current takes too long to follow the high-frequency oscillation of the impulse response. This results in a sawtooth shaped waveform as shown in Figure 2.9 which is the zoom-in part of Figure 2.8 around $50\mu s$. From Figure 2.10 it can be seen that the majority of the current energy is actually at around the lowest resonant frequency as the oscillations corresponding to the middle and high resonant frequency die down much faster.

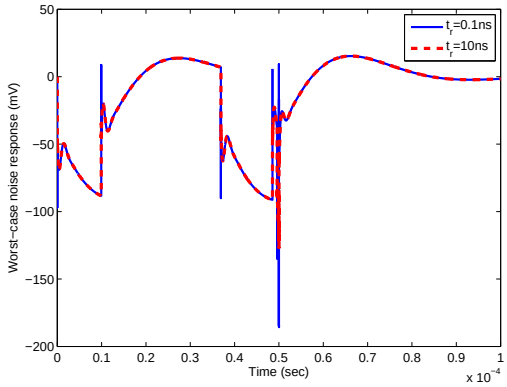


Figure 2.7: Worst-case noise responses of the power distribution system at $t_r = 100ps$ and $t_r = 10ns$.

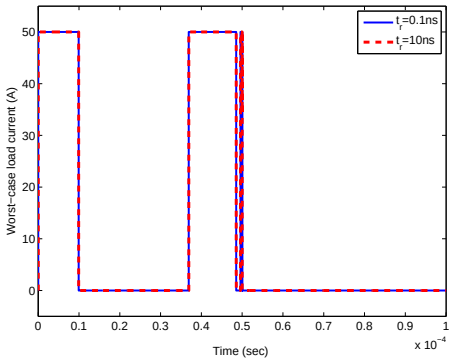


Figure 2.8: Worst-case load currents at $t_r = 100ps$ and $t_r = 10ns$.

The maximum voltage drop is plotted as a function of transition time in Figure 2.11. The worst-case noise is a monotonically decreasing function of tran-

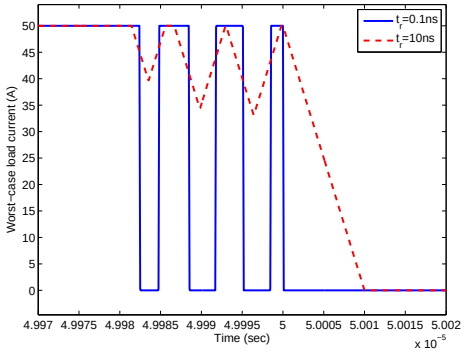


Figure 2.9: Zoom-in part of the worst-case load currents at $t_r = 100ps$ and $t_r = 10ns$ around $50\mu s$.

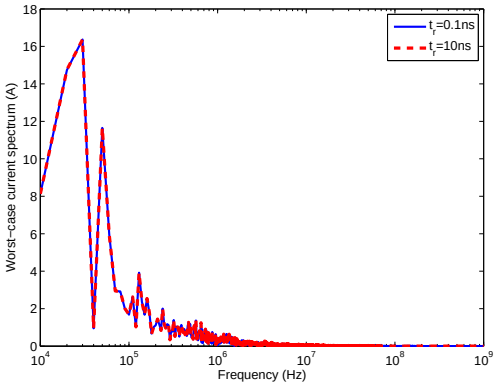


Figure 2.10: Spectrum of the worst-case load currents at $t_r = 100ps$ and $t_r = 10ns$.

sition time. This can be explained from our problem formulation: the smaller the transition time is, the more chances there are that the positive part of the impulse response is multiplied by a large value of current or vice versa. When the transition time is zero, our results match those in [HZZ+09] and the worst-case noise is the largest. This result demonstrates that assuming a zero transition time will result in a worst-case noise that is overly pessimistic.

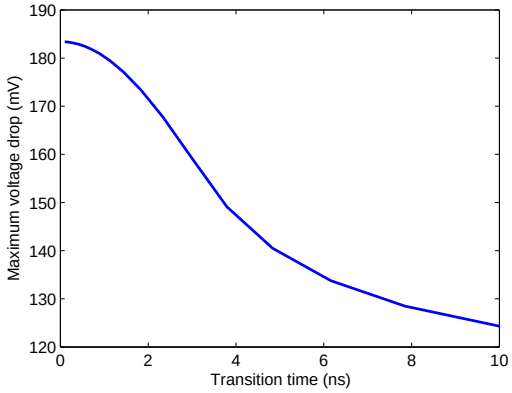


Figure 2.11: Worst-case voltage drop as a function of transition time.

The run time of the proposed algorithm is tested in this sub-section. The algorithm has been implemented in C++ and is tested in a Pentium(R) 4 3.00GHz computer with 1.00GB memory. The number of sample points of the impulse response $z(t)$ is one million and the number of zero-crossing points m equals to 12. Figure 2.12 compares the run time of the original and accelerated algorithm as a function of the number of discretized current values n. The run time follows the n^2m and $nm \log n$ trends for the original and accelerated algorithms, respectively, as discussed in the above sections. The run time of the accelerated algorithm is much smaller than that of the original one, especially when n is large.

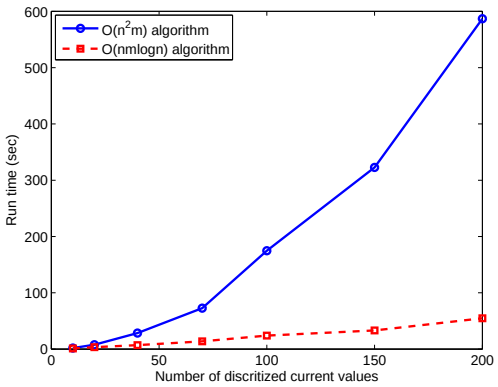


Figure 2.12: Run time comparison of $O(n^2)$ and $O(n \log n)$ algorithm.

2.2 Multiple Current Sources

2.2.1 Problem Formulation

Our aim is to identify the worst-case noise under the current constraints. As discussed in section 2.2.2, one class of the constraints on the load currents $\vec{I}(t) = (I_1(t), I_2(t), \ldots, I_n(t))^T$ regards the amplitude, which can be expressed as:

$$A\vec{I}(t) \leq \vec{B}, \tag{2.9}$$

$$I_k(t) \geq 0, \;\; 0 \leq k \leq n, \tag{2.10}$$

where A is an $m \times n$ matrix satisfying HCC condition and $\vec{B} = (B_1, B_2, \ldots, B_m)$ is a vector of constant upper bounds. Since the transition time of the total load current is assumed to be non-zero, we add another constraint for the slope of the total load current as follows,

$$-C \leq \frac{d\sum_{k=1}^{n} I_k(t)}{dt} \leq C, \;\; \forall t \geq 0, \tag{2.11}$$

where C is a constant upper bound.

We denote the noise of the power grid by $v(t)$. For noise analysis, the voltage source V_{dd} is considered as zero. Thus, $v(t)$ can be evaluated by the summation of convoluting the input currents with impulse responses as:

$$v(t) = \sum_{k=1}^{n} \int_0^t Z_k(\tau) I_k(t-\tau)\, d\tau, \tag{2.12}$$

where $(Z_1(t), Z_2(t), \ldots, Z_n(t))$ are the impulse responses of the power grid. Since the convolution operation has an accumulative effect on $v(t)$, we set the integration time, which is denoted by T, to be such that the impulse responses have died down to some negligible value. Now we can formulate the problem of finding the worst-case noise as:

$$\begin{aligned} \max \quad & v(T) = \sum_{k=1}^{n} \int_0^T Z_k(t) I_k(t)\, dt \\ \text{s.t.} \quad & A\vec{I}(t) \leq \vec{B}, \\ & I_k(t) \geq 0, \;\; 1 \leq k \leq n, \\ & -C \leq \frac{d\sum_{k=1}^{n} I_k(t)}{dt} \leq C, \;\; \forall t \geq 0, \end{aligned} \tag{2.13}$$

where we replace $I_k(T-t)$ with $I_k(t)$ since it has no effect on this optimization problem.

2.2.2 Hierarchical Current Constraints

Suppose our on-chip power grid has n nonnegative load currents which are denoted by $I_1(t)$,$I_2(t)$,...,$I_n(t)$. Given a fixed time point t_0, we denote $I_k(t_0)$ by I_k where $1 \leq k \leq n$ for convenience. If S is a subset of $\{1, 2, \ldots, n\}$ and B is a constant upper bound, we define a constraint on load currents corresponding to S and B to be $L(S, B) \doteq \sum_{k \in S} I_k \leq B$. A Hierarchical Current Constraints (HCC) is a collection of constraints $L(S_1, B_1), L(S_2, B_2), \ldots, L(S_m, B_m)$ where one of $S_i \cap S_j = \emptyset$, $S_i \subset S_j$ and $S_j \subset S_i$ holds for each pair of S_i and S_j. An example of HCC is showed in Table 2.2.

Table 2.2: An HCC example with seven load currents and twelve constraints

Notation	Constraint
$L(\{1\}, B_1), \ldots, L(\{7\}, B_7)$	$I_1 \leq B_1, \ldots, I_7 \leq B_7$
$L(\{1, 2\}, B_8)$	$I_1 + I_2 \leq B_8$
$L(\{3, 4, 5\}, B_9)$	$I_3 + I_4 + I_5 \leq B_9$
$L(\{1, 2, 3, 4, 5\}, B_{10})$	$I_1 + I_2 + I_3 + I_4 + I_5 \leq B_{10}$
$L(\{6, 7\}, B_{11})$	$I_6 + I_7 \leq B_{11}$
$L(\{1, 2, \ldots, 7\}, B_{12})$	$I_1 + I_2 + \cdots + I_7 \leq B_{12}$

In the following, we will give several properties of the set of load currents constrained by HCC. These properties lead to our efficient algorithm for worst case generation.

Definition 1. *Since HCC is a special class of linear constraints, we say that an $m \times n$ matrix A satisfies* HCC condition *if and only if $A\vec{I} \leq \vec{B}$ is an HCC where $\vec{I}$ denotes $(I_1, I_2, \ldots, I_n)^T$ and $\vec{B}$ denotes $(B_1, B_2, \ldots, B_m)^T$.*

Definition 2. *A* maximal feasible solution *for an HCC is a feasible solution $(I_1^f, I_2^f, \ldots, I_n^f)$ with the property that there does not exist $\epsilon > 0$ and $1 \leq k \leq n$ such that $(I_1^f, I_2^f, \ldots, I_k^f + \epsilon, \ldots, I_n^f)$ is feasible.*

Lemma 6. *Suppose* $\vec{I}^{f_1} = (I_1^{f_1}, I_2^{f_1}, \ldots, I_n^{f_1})$ *and* $\vec{I}^{f_2} = (I_1^{f_2}, I_2^{f_2}, \ldots, I_n^{f_2})$ *are two maximal feasible solutions for an HCC, we have* $\sum_{k=1}^{n} I_k^{f_1} = \sum_{k=1}^{n} I_k^{f_2}$. *This value will be called the* feasible upper bound *of the HCC.*

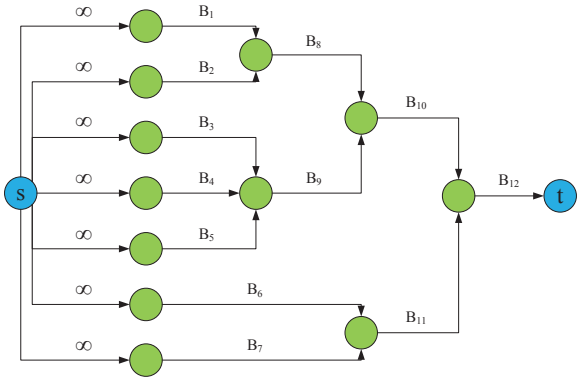


Figure 2.13: The flow network for the HCC in Table 2.2.

Proof. We construct a corresponding flow network G for the HCC as showed in Figure 2.13. Each constraint is represented by a node and two extra nodes s, t are added to the network for source and sink respectively. For each pair of constraints $L(S_1, B_1)$ and $L(S_2, B_2)$, if $S_1 \subset S_2$ and there is no constraint $L(S_3, B_3)$ such that $S_1 \subset S_3 \subset S_2$, we add an edge in G from the node for $L(S_1, B_1)$ to the node for $L(S_2, B_2)$ with capacity B_1. In other words, G without s and t has the same topological structure as the Hasse diagram of the partial order set $(\mathcal{S}, \subseteq)$ where $\mathcal{S}$ denotes the collection of all sets of load currents showed in the HCC and $\subseteq$ denotes the relation of set containment. We connect s to all minimal elements in the partial order set with capacity infinity. We also connect all maximal elements to t with capacity corresponding to the constraint upper bound for that element. Clearly, feasible solutions for the HCC have a one-to-one correspondence with feasible flows in network G. Therefore, based on Ford-Fulkerson algorithm [CLRS01], a maximal feasible solution corresponds to a maximum flow since the flow has no augmenting path. Because the value of the maximum flow is unique for the network G, we conclude that $\sum_{k=1}^{n} I_k^{f_1} = \sum_{k=1}^{n} I_k^{f_2}$. □

Given an HCC $L(S_1, B_1), L(S_2, B_2), \ldots, L(S_m, B_m)$ represented by $A\vec{I} \leq \vec{B}$,

we consider the following linear programming problem:

$$\begin{aligned} \max \quad & \vec{C}\vec{I} \\ \text{s.t.} \quad & A\vec{I} \leq \vec{B}, \\ & \sum_{k=1}^{n} I_k = D, \end{aligned} \tag{2.14}$$

where $\vec{C} = (c_1, c_2, \ldots, c_n)$ is a vector of constant coefficients and D is a positive constant. We denote the optimal solution for problem 2.14 by $\vec{I^*} = (I_1^*, I_2^*, \ldots, I_n^*)$ which can be found by the following greedy algorithm. First, we reorder the coordinates of $\vec{C}$ as $c_{i_1} \geq c_{i_2} \geq \cdots \geq c_{i_n}$. Suppose $I_{i_1}^*, \ldots, I_{i_k}^*$ are known, we choose $I_{i_{k+1}}^*$ to be the largest $\hat{I}$ such that $(I_{i_1}^*, \ldots, I_{i_k}^*, \hat{I}, 0, \ldots, 0)$ is a feasible solution. Notice that there is a solution for problem 2.14 if and only if D is less than or equal to the feasible upper bound of $A\vec{I} \leq \vec{B}$. Moreover, the optimal solution $\vec{I^*}$ only depends on the order $(i_1, i_2, \ldots, i_n)$ of the coordinates of $\vec{C}$ instead of the values of them. Therefore, we can define a function $\vec{I^*} = GetOptCurrents(A, \vec{B}, D, \vec{P})$ where $\vec{P} = (i_1, i_2, \ldots, i_n)$ gives the order of the coordinates of $\vec{C}$.

Theorem 4. *The load currents $\vec{I^*}$ computed by the greedy algorithm is an optimal solution for problem 2.14.*

Proof. At first, we reset the indices of the inputs so that $c_1 \geq c_2 \geq \cdots \geq c_n$. Let $(I_1^*, \ldots, I_n^*)$ be the feasible solution computed by the greedy algorithm and $(\tilde{I}_1, \ldots, \tilde{I}_n)$ be an optimal solution. Suppose k is the first place where these two solutions are different, i.e. $\tilde{I}_1 = I_1^*, \ldots, \tilde{I}_{k-1} = I_{k-1}^*$ and $\tilde{I}_k < I_k^*$. Let $\mathcal{S} = \{S_{i_1}, S_{i_2}, \ldots, S_{i_t}\}$ be the collection of sets containing k. Since the joint of any two sets in $\mathcal{S}$ is non-empty, we can assume $S_{i_1} \subset S_{i_2} \subset \cdots \subset S_{i_t}$ based on the definition of HCC. Suppose S_{i_p} is the first set in the sequence $S_{i_1}, S_{i_2} \ldots, S_{i_t}$ containing an index k' such that $k' > k$ and $\tilde{I}_{k'} > 0$. If no such S_{i_p} exists, we take k' to be any index satisfying $k' > k$ and $\tilde{I}_{k'} > 0$ where k' exists since $\sum_{q=1}^{n} \tilde{I}_q = \sum_{q=1}^{n} I_q^* = D$ and $\tilde{I}_k < I_k^*$. Let $\delta = \min\{I_k^* - \tilde{I}_k, \tilde{I}_{k'}\}$. We can construct a new optimal solution by changing $\tilde{I}_k$ to $\tilde{I}_k + \delta$ and changing $\tilde{I}_{k'}$ to $\tilde{I}_{k'} - \delta$ in $(\tilde{I}_1, \ldots, \tilde{I}_n)$. By doing this process finite times, we can form an optimal solution with the first k coordinates equal to the greedy one. Finally, we make the conclusion that $(I_1^*, \ldots, I_n^*)$ is optimal by induction. □

2.2.3 Linear Programming on Submodular Polyhedron

In this section, we describe the general theoretical framework in [Fuj05] behind the hierarchical constraint and the greedy algorithm presented in the previous section. Through this framework, we can generalize our worst-case generation algorithms in subsequent sections from hierarchical constraints to submodular polyhedron. We will build the framework in three steps: (1). Give the definitions of the submodular system and submodular polyhedron, and show hierarchical constraints can be extended into a submodular polyhedron; (2). Present the formulation of linear programming on submodular polyhedron and the optimality condition for it; (3). Describe the greedy algorithm for solving the linear programming problem. In the whole section, we use E to denote a nonempty finite set called ground set and define $\mathscr{D}$ to be a family of subsets of E forming a distributive lattice with set union and intersection as join and meet of lattice operations. Clearly, $\mathscr{D}$ need to be closed under the set union and intersection.

Definition 3. *A function $f : \mathscr{D} \to \mathbb{R}$ is submodular on the distributive lattice $\mathscr{D}$ if and only if*

$$f(X \cup Y) + f(X \cap Y) \leq f(X) + f(Y), \quad \forall X, Y \in \mathscr{D}$$

Definition 4. *We call a pair $(\mathscr{D}, f)$ a submodular system on E if*

- *$\mathscr{D}$ is a distributive lattice on E with $\emptyset \in \mathscr{D}$ and $E \in \mathscr{D}$;*
- *f is a submodular function on D with $f(\emptyset) = 0$;*

In this case, we call f the rank function of the submodular system.

Definition 5. *We define $\mathbb{R}^E$ to be the function space containing all functions mapping E to $\mathbb{R}$. For any $x \in \mathbb{R}^E$ and $X \in \mathscr{D}$, we define $x(X) = \sum_{e \in X} x(e)$.*

Definition 6. *We define the submodular polyhedron $P(f)$ in $\mathbb{R}^E$ associated with the submodular system $(\mathscr{D}, f)$ by*

$$P(f) = \{x \in \mathbb{R}^E : x(X) \leq f(X),\ \forall X \in \mathscr{D}\}$$

Similarly, we define the base polyhedron associated with $(\mathscr{D}, f)$ *by*

$$B(f) = \{x \in P(f),\ x(E) = f(E)\}$$

We notice that a hierarchical family of subsets of E does not naturally form a distributive lattice on E since it is not closed on set union. However, a real-valued function f on a hierarchical family can be extended to a submodular function on a distributive lattice and the polyhedron determined by the corresponding hierarchical constraints is a submodular polyhedron.

Definition 7. *A family* $\mathscr{F}$ *of subsets of* E *is called an intersecting family if for each intersecting* $X, Y \in \mathscr{F}$ *(i.e.,* $X \cap Y \neq \emptyset$*), we have* $X \cup Y \in \mathscr{F}$ *and* $X \cap Y \in \mathscr{F}$*. A function* $f : \mathscr{F} \to \mathbb{R}$ *on the intersection family mathscrF is called intersecting-submodular if for any intersecting* $X, Y \in \mathscr{F}$*, we have the submodularity inequality:*

$$f(X \cup Y) + f(X \cap Y) \leq f(X) + f(Y)$$

Lemma 7. *A hierarchical family* $\mathscr{F}$ *is an intersection family and any real-valued function* f *on* $\mathscr{F}$ *is intersection-submodular.*

Proof. For any intersecting $X, Y \in \mathscr{F}$, we have $X \subseteq Y$ or $Y \subseteq X$ by the definition of hierarchical family. Hence $X \cup Y$ and $X \cap Y$ equal to X or Y which belongs to $\mathscr{F}$. We conclude that $\mathscr{F}$ is an intersection family. Moreover, any real-valued function f on $\mathscr{F}$ satisfies $f(X \cup Y) + f(X \cap Y) = f(X) + f(Y)$ which shows f is intersection-submodular. □

The following important theorem shown in [Fuj05] gives a claim that the polyhedron determined by a intersection-submodular function is actually a submodular polyhedron. In particular, the polyhedron determined by hierarchical constraints is a submodular polyhedron.

Theorem 5. *Let* $\mathscr{F}$ *be an intersection family with* $\emptyset \in \mathscr{F}$ *and* $E \in \mathscr{F}$*. Let* f *be an intersecting-submodular function on* $\mathscr{F}$*. We define the polyhedron determined by* f *as*

$$P(f) = \{x \in \mathbb{R}^E : x(X) \leq f(X),\ \forall X \in \mathscr{F}\}$$

Then there exists a unique submodular system $(\mathscr{D}_1, f_1)$ *on* E *such that*

$$P(f) = P(f_1)$$

Moreover, if f *is integer-valued, then* f_1 *is also integer-valued.*

Now we consider the following linear programming problem for a submodular system $(\mathscr{D}, f)$ on E:

$$\begin{aligned} \min \quad & \sum_{e \in E} w(e)x(e) \\ \text{s.t.} \quad & x \in B(f), \end{aligned} \tag{2.15}$$

where $w : E \to \mathbb{R}$ is a given weight function. We derive the condition for the existence of a finite optimal solution for 2.15 before we give the greedy algorithm for solving it.

Based on the partial order set theory, for each distributive lattice $\mathscr{D}$ on E with $\emptyset, E \in \mathscr{D}$, we can construct a partial order set $P(\mathscr{D})$ on partitions of E to represent it. We describe the detail of the process as follows:

Definition 8. *A chain* C *of* $\mathscr{D}$ *is defined as a sequence of monotone increasing elements of* $\mathscr{D}$*, i.e.*

$$C : S_0 \subset S_1 \subset \cdots \subset S_k$$

C *is called a maximal chain if there is no chain containing* C *as a proper subsequence. Clearly, we have* $S_0 = \emptyset$ *and* $S_k = E$ *for a maximal chain* C*.*

Definition 9. *For each* $e \in E$*, we define* $\mathscr{D}(e)$ *as the unique minimal element of* $\mathscr{D}$ *containing* e*, i.e.*

$$\mathscr{D}(e) = \bigcap \{X \in \mathscr{D} : e \in X\}$$

Definition 10. *We define* $G(\mathscr{D}) = (E, A)$ *to be the directed graph on vertex set* E *with arc set*

$$A = \{(e, e') : e \in E, e' \in \mathscr{D}(e)\}.$$

If we decompose the graph $G(\mathscr{D})$ into strongly connected components $G_1, \ldots, G_k \subseteq E$ which forms a partition of the vertex set E, then there exists a natural partial

order $\preceq_{\mathscr{D}}$ on $\{G_1, G_2, \ldots, G_k\}$ where $G_i \preceq_{\mathscr{D}} G_j$ if and only if there is a directed path from a vertex of G_j to a vertex of G_i. Let $\Pi(\mathscr{D}) = \{G_1, G_2, \ldots, G_k\}$ and we define $P(\mathscr{D}) = (\Pi(\mathscr{D}), \preceq_{\mathscr{D}})$ as the *poset derived from distributive lattice* $\mathscr{D}$. The following two theorems characterize the one to one relationship between a distributive lattice $\mathscr{D}$ and the poset $P(\mathscr{D})$ derived from it.

Definition 11. *For a poset $\mathscr{P} = (P, \preceq)$, a set $J \subseteq P$ is called an ideal of $\mathscr{P}$ if $e \in J$ and $e' \preceq e$ implies that $e' \in J$.*

Theorem 6. *Let $\mathscr{D}$ be a distributive lattice on E with $\emptyset, E \in \mathscr{D}$ and $P(\mathscr{D})$ be the poset derived from $\mathscr{D}$. Then the following two properties hold:*

- *For any ideal J of $P(\mathscr{D})$, we have*

$$\bigcup_{F \in J} F \in \mathscr{D}.$$

- *For any $X \in \mathscr{D}$, there exists an ideal J of $P(\mathscr{D})$ such that*

$$X = \bigcup_{F \in J} F.$$

The two properties give the one to one correspondence between the elements of $\mathscr{D}$ and ideals of $P(\mathscr{D})$. Conversely, for any poset $\mathscr{P} = (P, \preceq)$ on a partition P of E, we can obtain a distributive lattice $\mathscr{D}$ defined by

$$\mathscr{D} = \{\bigcup_{F \in J} F : J \text{ is an ideal of } \mathscr{P}\}$$

Therefore, the construction is two-directional and there is a one to one mapping between the set of distributive lattice $\mathscr{D}$ on E with $\emptyset, E \in \mathscr{D}$ and that of posets $\mathscr{P} = (P, \preceq)$ on partitions P of E.

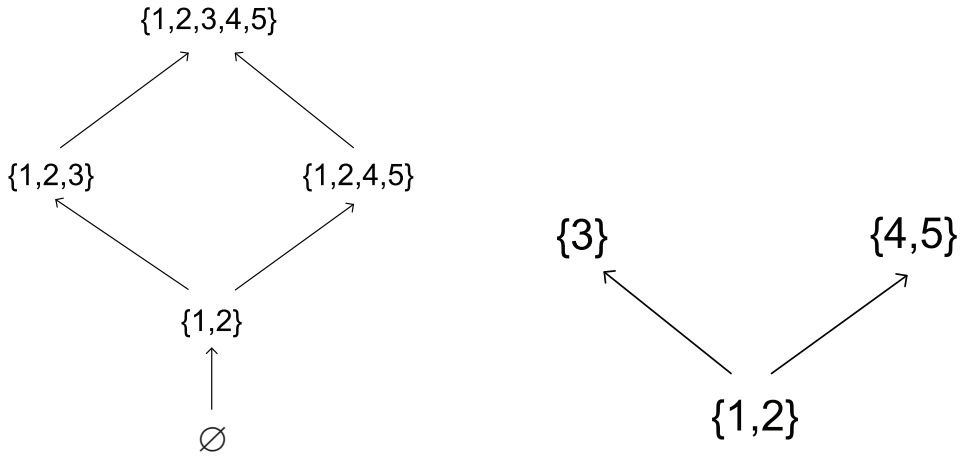
Theorem 7. *For a distributive lattice $\mathscr{D}$ on E with $\emptyset, E \in \mathscr{D}$, let*

$$C : \emptyset = S_0 \subset S_1 \subset \cdots \subset S_k = E$$

be an arbitrary maximal chain of mathscr$D$. Then we have

$$\Pi(\mathscr{D}) = \{S_i - S_{i-1} : i = 1, 2, \ldots, k\}.$$

In particular, the lengths of all maximal chains of $\mathscr{D}$ are equal to $|\Pi(\mathscr{D})|$.



(a) A distributive lattice $\mathscr{D}$. (b) The corresponding $P(\mathscr{D})$.

Figure 2.14: An example for the relationship between $\mathscr{D}$ and $P(\mathscr{D})$.

Figure 2.14 gives an example for a distributive lattice $\mathscr{D}$ and its corresponding poset $P(\mathscr{D})$ on a partition of E. The poset $P(\mathscr{D})$ contains five ideals $\emptyset$,$\{\{1,2\}\}$,$\{\{1,2\},\{3\}\}$,$\{\{1,2\},\{4,5\}\}$ and $\{\{1,2\},\{3\},\{4,5\}\}$ which corresponds to the elements $\emptyset$,$\{1,2\}$,$\{1,2,3\}$,$\{1,2,4,5\}$ and $\{1,2,3,4,5\}$ in $\mathscr{D}$. There are two maximal chains $\emptyset \subset \{1,2\} \subset \{1,2,3\} \subset \{1,2,3,4,5\}$ and $\emptyset \subset \{1,2\} \subset \{1,2,4,5\} \subset \{1,2,3,4,5\}$ in the distributive lattice $\mathscr{D}$ where the collection of the difference of adjacent elements for both chains is $\{1,2\}$,$\{3\}$ and $\{4,5\}$, i.e. the element set of $P(\mathscr{D})$. Now we are ready to give the optimality condition of the linear programming formulation 2.15 by the following theorem:

Theorem 8. *The linear programming 2.15 has a finite optimal solution if and only if the weight function w satisfies the following two properties:*

- *(1): For any two elements e_1 and e_2 in E with $e_1, e_2 \in S$ for some $S \in P(\mathscr{D})$, we have $w(e_1) = w(e_2)$.*
- *(2): For any two elements e_1 and e_2 in E with $e_1 \in S_1$, $e_2 \in S_2$ for some $S_1, S_2 \in P(\mathscr{D})$ satisfying $S_1 \preceq_{\mathscr{D}} S_2$, we have $w(e_1) \leq w(e_2)$.*

We call w is compatible with $P(\mathscr{D})$ if both properties hold.

Given a weight function w compatible with $P(\mathscr{D})$, we can find the function x achieving the optimal solution for 2.15 via the following greedy algorithm:

- Step one: Find a linear extension $(S_1, S_2, \ldots, S_k)$ of the poset $P(\mathscr{D})$ where k is the size of $\Pi(\mathscr{D})$.

- Step two: We define $T_0 = \emptyset$ and $T_i = S_i \cup T_{i-1}$ for $1 \leq i \leq k$. For each element $e \in S_i$, we assign $x(e) = f(T_i) - f(T_{i-1})$. Notice that the set $\{S_1, S_2, \ldots, S_i\}$ is an ideal of $P(\mathscr{D})$ for any $1 \leq i \leq k$. Therefore T_i for any $0 \leq i \leq k$ is an element of $\mathscr{D}$ and $f(T_i)$ is well-defined.

2.2.4 Method one: Dynamic Programming

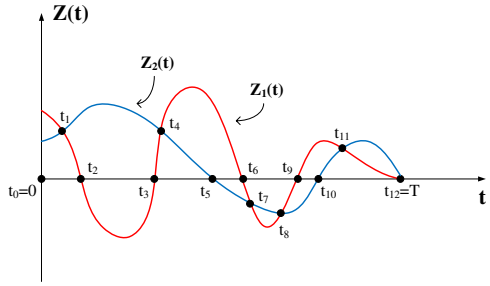
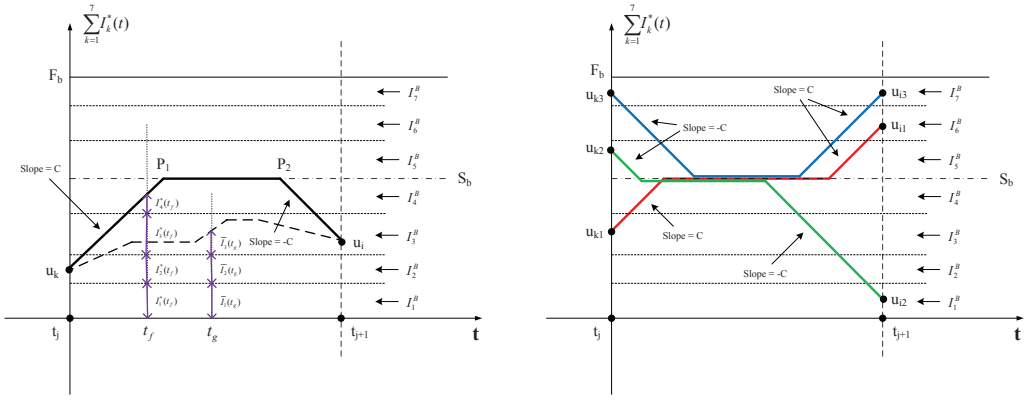


Figure 2.15: The division of $[0, T]$ into twelve intervals.

In order to solve the optimization problem (2.13), we first divide the time range $[0, T]$ into m intervals $[t_0 = 0, t_1], [t_1, t_2], \ldots, [t_{m-1}, t_m = T]$ where $t_1, \ldots, t_{m-1}$ are the positions in the time axis corresponding to either a zero-crossing point of some impulse response or the intersection point of two impulse responses. Figure 2.15 gives an example of two impulse responses divided into twelve time intervals. We also choose $s+1$ sample points $u_0 = 0, u_1, \ldots, u_{s-1}, u_s = F_b$ in the range of the current $[0, F_b]$ where s depends on the precision requirement and F_b is the feasible upper bound corresponding to the HCC $A\vec{I}(t) \leq \vec{B}$.

Definition 12. *Given a time interval $[t_j, t_{j+1}]$, we define the weight function $W_j(k, i)$ as the worst-case noise generated by the load currents starting with sum u_k at time t_j and ending with sum u_i at time t_{j+1}. The corresponding optimal load*



(a) The choice of optimal load currents. (b) Other cases for various endpoints.

Figure 2.16: The greedy sum of load currents in the time interval $[t_j, t_{j+1}]$.

currents are defined as $(I_1^*(t), I_2^*(t), \ldots, I_n^*(t))$ *where* $t \in [t_j, t_{j+1}]$ *and we have*

$$W_j(k,i) = \sum_{q=1}^{n} \int_{t_j}^{t_{j+1}} Z_q(t) I_q^*(t)\, dt \tag{2.16}$$

Let the optimal function $OPT(j,i)$ with $j \in [0,m]$ and $i \in [0,s]$ be the worst-case noise generated by the currents stopping at time t_j with sum u_i. The base case and recursive formula for the function OPT should be

$$\begin{aligned} OPT(0,i) &= 0 \quad \text{for all } i \in [0,s] \\ OPT(j+1,i) &= \max_{0 \le k \le n} (OPT(j,k) + W_j(k,i)) \end{aligned} \tag{2.17}$$

which gives us a dynamic programming algorithm to compute all values of the function OPT and the maximum $v(T)$ is the largest one in the set $\{OPT(m,i) : i \in [0,s]\}$. Notice that the dynamic programming algorithm only consider the time stages $0, t_1, t_2, \ldots, t_{m-1}, T$ instead of every sampled time points, which reduces the time complexity of the computation significantly.

We can choose the optimal load currents $(I_1^*(t), \ldots, I_n^*(t))$ in (2.16) satisfying constraints in (2.13) by a greedy algorithm. Since the order and sign of the impulse responses will not change in interval $[t_j, t_{j+1}]$, we reset the indices so that $Z_1(t) \ge \cdots \ge Z_p(t) \ge 0 \ge Z_{p+1}(t) \ge \cdots \ge Z_n(t)$ for convenience. Let $\vec{I}^B = (I_1^B, \ldots, I_n^B)$ be the solution of $GetOptCurrents(A, \vec{B}, F_b, \vec{P})$ where

$\vec{P} = (1, 2, \ldots, n)$. We define S_b as $\sum_{q=1}^{p} I_q^B$, i.e. the sum of elements in $\vec{I}^B$ corresponding to nonnegative impulse responses. As shown in Figure 2.16(a) for the sum of optimal currents where $n = 7$ and $p = 4$, both of u_k and u_i are less than S_b. In this case, we draw a line starting from u_k with slope H and another line ending at u_i with slope $-H$. Suppose these two lines intersect with S_b at points P_1 and P_2 respectively, we take the line segments $u_k \to P_1 \to P_2 \to u_i$ as the sum of optimal currents. Given a time point $t_f \in [t_j, t_{j+1}]$, by theorem 4, we set $I_5^*(t_f) = I_6^*(t_f) = I_7^*(t_f) = 0$ and choose $(I_1^*(t_f), \ldots, I_4^*(t_f))$ greedily from $(I_1^B, \ldots, I_4^B)$ until the sum is achieved. Other cases where the endpoints u_k and u_i are in various regions are showed in Figure 2.16(b). The red, green and blue line segments give the sum of optimal currents for $u_{k1} < S_b < u_{i1}$, $u_{k2} > S_b > u_{i2}$ and $u_{k3}, u_{i3} > S_b$ respectively. Once $(I_1^*(t), \ldots, I_n^*(t))$ are chosen, the weight function can be evaluated by equation (2.16). Note that if we choose any other non-greedy curve to be the sum of optimal currents as the dotted curve from u_k to u_i in Figure 2.16(a), the inequality $\tilde{I}_q(t) \leq I_q^*(t)$ will hold for any $t \in [t_j, t_{j+1}]$ and $q \in \{1, 2, \ldots, 7\}$ where $(\tilde{I}_1(t), \ldots, \tilde{I}_7(t))$ are chosen by the same greedy strategy on the non-greedy curve as $(I_1^*(t_f), \ldots, I_7^*(t_f))$. An example of assignment of $(\tilde{I}_1(t), \ldots, \tilde{I}_7(t))$ on a time point t_g is given in Figure 2.16(a). Let $W_j'(k, i) = \sum_{q=1}^{n} \int_{t_j}^{t_{j+1}} Z_q(t)\tilde{I}_q(t)\, dt$. Since $Z_1(t), \ldots, Z_4(t)$ are nonnegative in $[t_j, t_{j+1}]$, we have $W_j'(k, i) \leq W_j(k, i)$ which shows the correctness of our greedy choice of weight function.

If all values the OPT function are evaluated directly using the recursive formula, the time complexity is $O(s^2 m)$. In this section, we will use the modified Knuth-Yao quadrangle inequality speedup proposed in [DHW+10, Yao80] to accelerate our dynamic programming algorithm. It takes only $O(sm \log s)$ time to compute the OPT function.

We first fix j and rewrite the recursive formula as

$$OPT(j+1, i) = \max(OPT_1(j+1, i), OPT_2(j+1, i))$$

where

$$OPT_1(j+1, i) = \max_{0 \leq u_k \leq S_b} (OPT(j, k) + W_j(k, i))$$

and

$$OPT_2(j+1,i) = \max_{S_b < u_k \le F_b} (OPT(j,k) + W_j(k,i)).$$

Suppose $u_k > S_b$ if and only if $k \ge r$, we will give the speedup method only for $OPT_2(j+1,i)$ with $i \ge r$. Other cases for OPT_1 and OPT_2 can be computed similarly. The quadrangle inequality satisfied by the weight function is given as the following lemma:

Lemma 8. *$W_j(k_1,i_2) - W_j(k_2,i_2) \le W_j(k_1,i_1) - W_j(k_2,i_1)$ for any $r \le k_1 < k_2 \le s$ and $r \le i_1 < i_2 \le s$.*

proof 1. *As in Figure 2.17, $W_j(k_1,i_2)$ is the greedily chosen currents corresponding to the area below line segments $u_{k1} \to P_1 \to u_{i2}$ convoluted with $(Z_1(t),\ldots,Z_n(t))$. $W_j(k_2,i_2)$ is the greedily chosen currents corresponding to the area below line segments $u_{k2} \to Q_1 \to u_{i2}$ convoluted with $(Z_1(t),\ldots,Z_n(t))$. Therefore, the left hand side is the currents corresponding to the quadrangle area u_{k2}, Q_1, P_1, u_{k1} convoluted with $(|Z_1(t)|,\ldots,|Z_n(t)|)$. Similarly, the right hand side is the currents corresponding to the quadrangle area u_{k2}, Q_2, P_2, u_{k1} convoluted with $(|Z_1(t)|,\ldots,|Z_n(t)|)$. The inequality holds since the quadrangle u_{k2}, Q_2, P_2, u_{k1} contains u_{k2}, Q_1, P_1, u_{k1}.*

Definition 13. *We define $F_j(k,i)$ to be the candidate corresponding to k for $OPT(j+1,i)$, i.e. $F_j(k,i) = OPT(j,k) + W_j(k,i)$.*

Lemma 9. *Suppose $r \le k_1 < k_2 \le s$ and $r \le i_1 \le s$, $F_j(k_1,i_1) \le F_j(k_2,i_1)$ implies $F_j(k_1,i_2) \le F_j(k_2,i_2)$ for any $i_2 > i_1$.*

proof 2.

$$\begin{aligned}
& F_j(k_1,i_2) \\
= \;& OPT(j,k_1) + W_j(k_1,i_2) \\
= \;& (OPT(j,k_1) + W_j(k_1,i_1)) + (W_j(k_1,i_2) - W_j(k_1,i_1)) \\
= \;& F_j(k_1,i_1) + (W_j(k_1,i_2) - W_j(k_1,i_1)) \\
\le \;& F_j(k_2,i_1) + (W_j(k_1,i_2) - W_j(k_1,i_1)) \text{ by conditions} \\
\le \;& F_j(k_2,i_1) + (W_j(k_2,i_2) - W_j(k_2,i_1)) \text{ by lemma 8} \\
= \;& OPT(j,k_2) + W_j(k_2,i_2) \\
= \;& F_j(k_2,i_2)
\end{aligned}$$

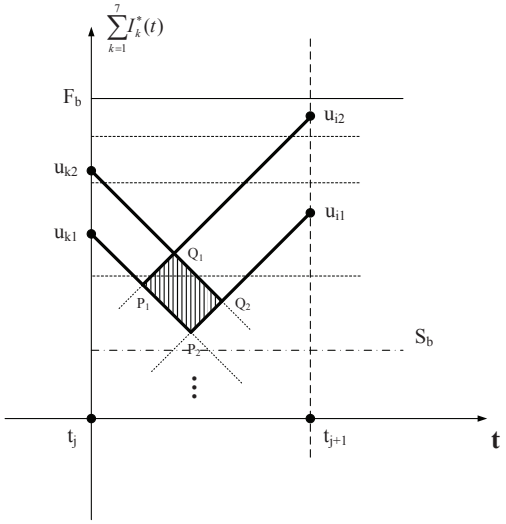


Figure 2.17: The quadrangle inequality for the weight function.

Lemma 9 shows that if k_2 is a better candidate than k_1 for $OPT_2(j+1, i_1)$, then for any $i_2 > i_1$, k_2 is still a better candidate. Therefore, we can find the smallest $i_0 \geq r$ such that $F_j(k_1, i) \leq F_j(k_2, i)$ whenever $i \geq i_0$ within $O(\log s)$ time by binary search. i_0 is called the transition position for candidates k_1 and k_2 and we will define a function $i_0 = GTP(j, k_1, k_2)$ where GTP stands for "Get Transition Position".

In order to compute $OPT_2(j+1, i)$ for all $r \leq i \leq s$, our algorithm iterates i from r to s and maintains a list L containing all available candidates k sorted from large to small on the value of $F_j(k, i)$. The algorithm also maintains a priority queue Q containing the transition position for every adjacent pair of candidates in L. Each element of Q have three fields, two for the candidates k_1, k_2 and another for the key $GTP(k_1, k_2)$. The priority queue supports three operations $GetMin, DelMin$ and $Insert$. $Q.GetMin()$ and $Q.DelMin()$ will return and delete the minimum element of the queue respectively. $Q.Insert(e)$ will insert the element e to the queue. As we know, all these operations for priority queue can be implemented in time $O(\log s)$. An example indicating the relation between L and Q is showed in Figure 2.18. Now we will describe our dynamic programming speedup by the following pseudocode which evaluates $OPT_2(j+1, i)$ for $r \leq i \leq s$ by $OPT(j, i)$ for $r \leq i \leq s$.

L=$\{r, r+1, \ldots, s\}$;
Q=$\{[GTP(j,k,k+1), k, k+1] : r \leq k \leq s-1\}$;
for $i = r, \ldots, s$
    $[i_0, k_1, k_2] = Q.GetMin()$;
    while $i_0 == i$
        $Q.DelMin()$;
        if k_1, k_2 are adjacent in L
            Delete k_1 in L;
            Find k_3 left adjacent to k_2 in L;
            $Q.Add([GTP(j,k_3,k_2), k_3, k_2])$;
        end
        $[i_0, k_1, k_2] = Q.GetMin()$;
    end
    q=The first element in L;
    $OPT_2(j+1, i) = OPT(j, q) + W_j(q, i)$;
end

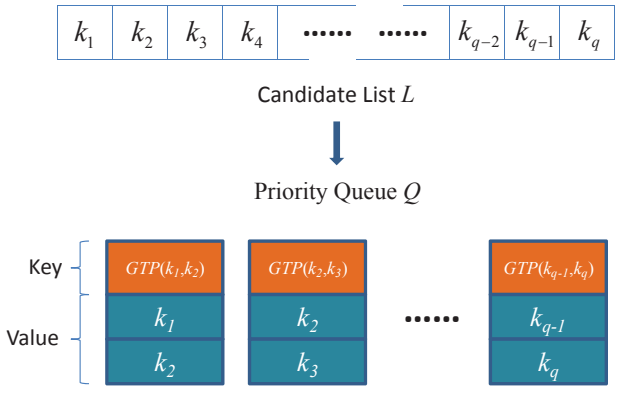


Figure 2.18: The relation between L and Q.

Theorem 9. *After the while loop of the pseudocode, L contains all possible candidates to compute $OPT_2(j+1, i)$ and $F_j(k_1, i) \geq F_j(k_2, i)$ for any $k_1, k_2 \in L$ and $k_1 < k_2$.*

Proof. In the *for* loop for a fixed i, we first retrieve the node $[i_0, k_1, k_2]$ from Q with minimum i_0. If i_0 is equal to i, we know that k_1 will not be a better candidate

than k_2 for the present and future i. Therefore, it is reasonable to delete k_1 in L. After the deletion, the element k_3 left adjacent to k_1 originally will be adjacent to k_2 in L now. To maintain the properties of Q, we add a new node for the adjacent pair k_3, k_2. Note that some adjacent pairs will be unavailable anymore during the deletion. For example, if k_1 has been deleted, the information recorded in Q for the pair k_3, k_1 becomes useless. If we come across such pairs in Q, i.e. the condition in the if line does not hold, we just skip the operations for deletion. After the *while* loop, we delete all impossible candidates from now on and the first claim holds. We also observe that if $k_1 < k_2$ and $F_j(k_1, i) < F_j(k_2, i)$, k_1 should be deleted in or before the iteration i, which shows the correctness of the second claim. □

Based on theorem 9, for each iteration of i, we can evaluate $OPT_2(j+1, i)$ as $OPT(j, q) + W_j(q, i)$ since the first element q in L is the best candidate.

Theorem 10. *The accelerated dynamic programming algorithm given by the pseudocode runs in time $O(s \log s)$.*

Proof. The candidate list L is implemented by a doubly-linked list in which the operations of deleting an element and finding the left adjacent element take constant time. We observe that each candidate k can only be deleted once, which implies that the pseudocode perform at most O(s) operations of the priority queue. Moreover, we call the GTP function at most $O(s)$ times, each of which takes $O(\log s)$ time. Therefore, the total time complexity of the pseudocode is $O(s \log s)$. □

We conclude that the OPT function can be computed within time $O(sm \log s)$ since we can initiate $OPT(0, i) = 0$ for all $i \in [0, s]$ and run the pseducode for $j = 1, 2, \ldots, m$ increasingly.

2.2.5 Method two: Network Flow with Side Constraints

In this section, we extend our formulation on worst case generation in two places: (1). Any group of current sources can have an upper bound instead of the case where groups form a hierarchical structure; (2). Each individual current source can have its own slope constraint rather than a slope constraint on the

sum of current sources. The new assumptions make the prediction more realistic. However, our original dynamic programming algorithm does not work for this case and we propose a network flow algorithm to generate the worst case noise. At first, the problem can be formally formulated as the following form:

$$\begin{aligned} \max \quad & v(T) = \sum_{k=1}^{n} \int_0^T Z_k(t) I_k(t)\, dt \\ \text{s.t.} \quad & A\vec{I}(t) \le \vec{B}, \\ & I_k(t) \ge 0, \;\; 1 \le k \le n, \\ & -C_k \le \frac{d(I_k(t))}{dt} \le C_k, \;\; 1 \le k \le n, \end{aligned} \tag{2.18}$$

where A is any $0-1$ matrix indicating groups of currents and C_k is the slope upper bound for the k^{th} current source. We split time range $[0, T]$ into m equal intervals $[t_0 = 0, t_1], [t_1, t_2], \ldots, [t_{m-1}, t_m]$ where $\Delta t = T/m$ and $t_i = i * \Delta t$. In each interval $[t_{i-1}, t_i]$, we approximate the impulse response $Z_k(t)$ on it by a constant denoted by $Z_k(i)$. Correspondingly, we represent the current source $I_k(t)$ by m discrete variables $I_k(1), I_k(2), \ldots, I_k(m)$. Now we can reformulate the problem as below:

$$\begin{aligned} \max \quad & v(T) = \sum_{k=1}^{n} \sum_{i=1}^{m} Z_k(i) I_k(i) \\ \text{s.t.} \quad & A\vec{I}(i) \le \vec{B}, \;\; 1 \le i \le m \\ & I_k(i) \ge 0, \;\; 1 \le k \le n, \, 1 \le i \le m \\ & -C_k \Delta t \le I_k(i) - I_k(i-1) \le C_k \Delta t, \;\; 1 \le k \le n, \, 1 \le i \le m, \end{aligned} \tag{2.19}$$

which is a standard linear programming problem which can be solved by libraries such as CPLEX and Gurobi. However, since the number of sample points need to be large enough in order to achieve acceptable precision, the time complexity of linear programming solver becomes infeasible for practical circuits. If we omit the group constraint $A\vec{I}(i) \le \vec{B}$ in 2.19, the problem can be simplified as a minimum-cost maximum-flow problem. Figure 2.19 shows the construction of the flow network with n current sources and m sample points on time range. We assume the current source $I_k(t)$ has an upper bound U_k which is determined by the group constraints. For each current source $I_k(t)$, a directed path $v_{k0} \to v_{k1} \to \cdots \to v_{km}$

is created in the network where the flow on the edge $v_{k(i-1)} \to v_{ki}$ corresponds to the variable $I_k(i)$, and the capacity and cost of the edge $v_{k(i-1)} \to v_{ki}$ is U_k and $Z_k(i)$ respectively. In order to represent the slope constraint for current sources, an auxiliary path $w_0 \to w_1 \to \cdots \to w_m$ is added to collect abundant flows and inject extra flows from/to other paths. We assign infinity and zero as the capacity and cost of an edge $w_{i-1} \to w_i$ for any $1 \leq i \leq m$. For each $1 \leq i \leq m-1$ and $1 \leq k \leq n$, two edges $w_i \to v_{ki}$ and $v_{ki} \to w_i$ are inserted into the network with capacity $C_k \Delta t$ and cost zero. We add two other extra nodes s and t indicating the source and the sink of the network. For each node u in $\{v_{k0} : 1 \leq k \leq n\} \cup \{w_0\}$, we put an edge $s \to u$ with capacity infinity and cost zero. For each node u in $\{v_{km} : 1 \leq k \leq n\} \cup \{w_m\}$, we an edge $u \to t$ with capacity infinity and cost zero. It is easy to see that a maximum cost flow from s to t corresponds to an optimal solution of the problem 2.19.

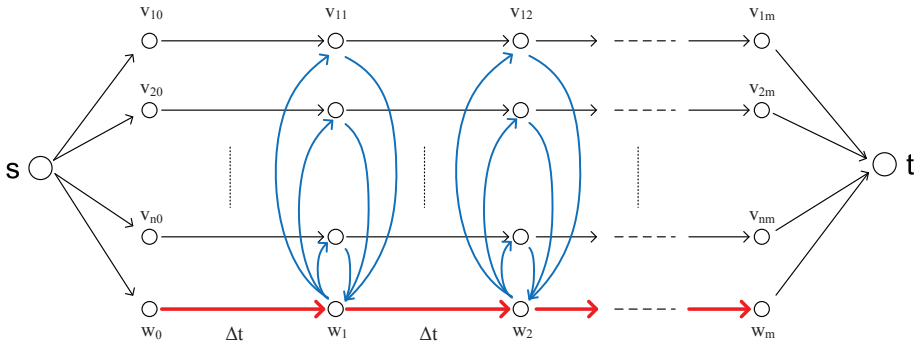


Figure 2.19: Flow network construction.

When we restrict that the current sources satisfies the group constraint $A\vec{I}(i) \leq \vec{B}$, we consider the following general minimum-cost maximum-flow problem with side constraints (Note that we can negate the edge costs in order to transform a maximum-cost flow problem into a minimum-cost problem):

$$\begin{aligned} \min \quad & \vec{w} \cdot \vec{x} \\ \text{s.t.} \quad & A\vec{x} \leq \vec{b}, \\ & G\vec{x} = \vec{d}, \\ & \vec{l} \leq \vec{x} \leq \vec{u}, \end{aligned} \tag{2.20}$$

We describe the notations in 2.20 as follows:

- $\vec{w}$: the cost function of edges;
- $\vec{x}$: the flow variables of edges;
- $A\vec{x} \leq \vec{b}$: the group constraint where A is a 0-1 matrix and $\vec{b}$ is a constant vector of upper bounds;
- $G\vec{x} = \vec{d}$: G is the incidence matrix of the network and d is the vector of supplies and demands;
- $\vec{l} \leq \vec{x} \leq \vec{u}$: l and u are constant vectors denoting the lower bound and upper bound of edge capacities;

Without the group (side) constraint $A\vec{x} \leq \vec{b}$ in 2.20, it reduces into a standard minimum-cost maximum-flow problem which can be solved efficiently. We define a vector set X to represent all feasible flow vector x satisfying flow conservation constraints, i.e.

$$X = \{\vec{x} : G\vec{x} = \vec{d} \text{ and } \vec{l} \leq \vec{x} \leq \vec{u}\}$$

Then the linear programming problem 2.20 can be rewritten as:

$$\begin{aligned} \min_{\vec{x} \in X} \quad & \vec{w} \cdot \vec{x} \\ \text{s.t.} \quad & A\vec{x} \leq \vec{b}, \end{aligned} \tag{2.21}$$

By the Lagrange duality theory introduced in [BV04a], we define the Lagrange dual function of the optimization problem 2.21 as:

$$L(\vec{\mu}, \vec{x}) = \vec{w} \cdot \vec{x} + \vec{\mu} \cdot (A\vec{x} - \vec{b}), \quad \vec{\mu} \geq 0 \,, \vec{x} \in X$$

Notice that $\max_{\vec{\mu}} L(\vec{\mu}, \vec{x})$ equals to $\vec{w} \cdot \vec{x}$ if $\vec{x}$ satisfies $A\vec{x} \leq \vec{b}$, and positive infinity otherwise. We conclude that the problem can be written as:

$$\min_{\vec{x} \in X} \max_{\vec{\mu} \geq 0} L(\vec{\mu}, \vec{x})$$

Since the strong duality property holds for linear programming problems, we can change the order of min and max in the previous formula. We define a function $g(\vec{\mu}) = \min_{\vec{x} \in X} L(\vec{\mu}, \vec{x})$ and obtain an equivalent optimization problem as below:

$$\max_{\vec{\mu} \geq 0} g(\vec{\mu}) = \min_{\vec{x} \in X} \max_{\vec{\mu} \geq 0} L(\vec{\mu}, \vec{x})$$

Lemma 10. *The function $g(\vec{\mu})$ is a concave function on $\vec{\mu}$.*

Proof. For each fixed $\vec{x}$, $L(\vec{\mu}, \vec{x})$ is an affine function on $\vec{\mu}$ which is both convex and concave. Therefore, $g(\vec{\mu})$ is concave since it is the pointwise infimum of a family of concave functions. □

Lemma 11. *For a fixed $\vec{\mu}$, $g(\vec{\mu})$ can be evaluated by the minimum-cost maximum-flow algorithm.*

Proof. By definition, $g(\vec{\mu})$ is the solution of the following optimization problem:

$$\begin{aligned} \min_{\vec{x}} \quad & \vec{w} \cdot \vec{x} + \vec{\mu}(A\vec{x} - \vec{b}) \\ \text{s.t.} \quad & G\vec{x} = \vec{d}, \\ & \vec{l} \leq \vec{x} \leq \vec{u}, \end{aligned} \tag{2.22}$$

which is a minimum-cost maximum-flow problem since the objective function is a linear function on $\vec{x}$ and the constraints contain only the flow conservation, plus lower and upper bounds of flow variables. □

By the lemmas 10 and 11, the problem of minimizing $g(\vec{\mu})$ becomes a convex programming problem which can be solved by the subgradient method introduced in [AMO93]. For the concave function $g(\vec{\mu})$, a vector $\vec{v}$ is called a *subgradient* of $g(\vec{\mu})$ at a fixed vector $\vec{\mu_0}$) if for any vector $\vec{\mu}$, we have $g(\vec{\mu}) - g(\vec{\mu_0}) \leq \vec{v} \cdot (\vec{\mu} - \vec{\mu_0})$. The following lemma gives an explicit formula for a subgradient vector of $g(\vec{\mu})$ at $\vec{\mu_0}$).

Lemma 12. *For a fixed vector $\vec{\mu_0}$, let $\vec{x}^*$ be a vector satisfying $g(\vec{\mu_0}) = \vec{w} \cdot \vec{x}^* + \vec{\mu_0}(A\vec{x}^* - \vec{b})$. Then $\vec{v} = A\vec{x}^* - \vec{b}$ is a subgradient vector of $g(\vec{\mu})$ at $\vec{\mu_0}$.*

Proof.

$$\begin{aligned} & g(\vec{\mu}) - g(\vec{\mu_0})) \\ \leq\ & (\vec{w} \cdot \vec{x^*} + \vec{\mu}(A\vec{x^*} - \vec{b})) - g(\vec{\mu_0}) \\ =\ & \vec{\mu}(A\vec{x^*} - \vec{b}) - \vec{\mu_0}(A\vec{x^*} - \vec{b}) \\ =\ & \vec{v}(\vec{\mu} - \vec{\mu_0}) \end{aligned}$$

□

By a similar iterative process as gradient decent method, we can optimize $g(\vec{\mu})$ via the pseudocode as follows:

Set a random initial vector $\vec{\mu}_0$;
$k = 0$;
while $\theta_k > \epsilon$
    Evaluate $g(\vec{\mu}_k)$ by the minimum-cost maximum-flow algorithm;
    Let $\vec{x^*}$ be a vector satisfying $g(\vec{\mu}_k) = \vec{w} \cdot \vec{x^*} + \vec{\mu}_k(A\vec{x^*} - \vec{b})$;
    $\vec{v} = A\vec{x^*} - \vec{b}$;
    $\vec{\mu}_{k+1} = \vec{\mu}_k + \theta_k \vec{v}$;
    $k = k + 1$;
end
return $g(\vec{\mu}_k)$**;**

where ϵ is the error tolerant constant and $(\theta_0, \theta_1, \ldots, \theta_k, \ldots)$ is a predefined sequence satisfying $\theta_k \to 0$ and $\sum_{j=1}^{k} \theta_j \to \infty$ as $k \to \infty$. A valid sequence of θ_k can be obtained by defining $\theta_k = 1/(k+1)$. The book [AMO93] introduces more algorithms for solving minimum-cost maximum-flow problem with side constraints such as *Dantzig-Wolfe decomposition or generalized linear programming*, and the simplex method with an advanced starting basis technique by Bixby.

2.2.6 Method three: Submodular Flow

In this section, we consider the worst-case generation problem with more constraints on the slopes of groups of current inputs with hierarchical structure.

We use the same notations and samples of range $[0, T]$ as the previous section. The modified problem formulation is described as follows:

$$\begin{aligned} \max \quad & v(T) = \sum_{k=1}^{n} \sum_{i=1}^{m} Z_k(i) I_k(i) \\ \text{s.t.} \quad & A\vec{I}(i) \leq \vec{B}, \;\; 1 \leq i \leq m \\ & I_k(i) \geq 0, \;\; 1 \leq k \leq n, \, 1 \leq i \leq m \\ & \sum_{k \in F} (I_k(i) - I_k(i-1)) \leq C(F) \Delta t, \\ & F \in \mathscr{F}, \, 1 \leq i \leq m, \end{aligned} \tag{2.23}$$

where $\mathscr{F}$ is a hierarchical family of subsets of $\{1, 2, \ldots, n\}$ and $C(F)$ is a non-negative valued function on $\mathscr{F}$. We utilize the same flow network construction in Figure 2.19 without the auxiliary path and omit the group constraint $A\vec{I}(i) \leq \vec{B}$ at first. Then we can reduce the original problem as the following minimum-cost maximum-flow problem with an extra constraint on the vector of supplies and demands corresponding to the slope constraints.

$$\begin{aligned} \min \quad & \vec{w} \cdot \vec{x} \\ \text{s.t.} \quad & G\vec{x} = \vec{d}, \\ & \vec{l} \leq \vec{x} \leq \vec{u}, \\ & \sum_{k \in F} \vec{d}_k \leq C'(F), \;\; F \in \mathscr{F}', \end{aligned} \tag{2.24}$$

where $\vec{w}, \vec{x}, G, \vec{l}, \vec{u}$ have the same meaning as 2.20, $\vec{d}$ becomes a variable rather than constant vector in 2.20, $\mathscr{F}'$ is a hierarchical family of subsets of the network's vertex set determined by the family $\mathscr{F}$ and the function $C'(F)$ is directly induced by the function $C(F)$. Based on the discussion in section 2.2.3, the constraints $\sum_{k \in F} \vec{d}_k \leq C'(F)$ can be treated as $\vec{d}$ belongs to a submodular polyhedron. Therefore, the formulation 2.24 is a standard submodular flow problem which can be solved efficiently as described in [BV04a]. In order to embed the group constraint $A\vec{I}(i) \leq \vec{B}$, we can use the same technique via Lagrange duality and solve the whole problem by combining subgradient method with submodular flow algorithm.

2.2.7 Experimental Results and Summary

We implemented our algorithm using C/C++ on a 64-bit linux machine with a 3.0GHz Intel Xeon processor. An industrial power grid with size $0.4mm$ by $0.4mm$ and four metal layers are considered in our experiment. n ports are uniformly chosen across the power grid and the impulse response of each port is generated by SPICE simulation.

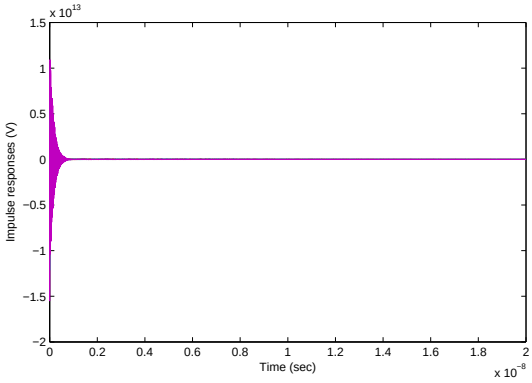


Figure 2.20: The impulse responses of the power grid.

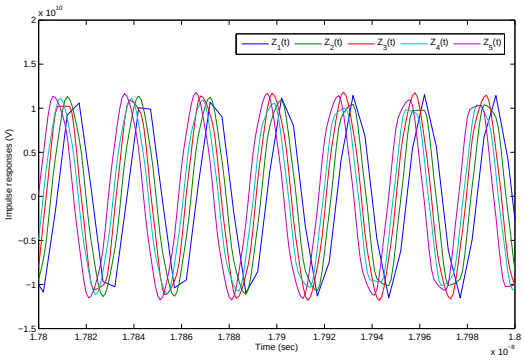


Figure 2.21: Zoom-in part of the impulse responses.

In order to demonstrate the effectiveness of our algorithm clearly, we choose a situation with $n = 5$. The impulse responses of the five ports are shown in Figure 2.20 with $T = 20ns$, and a close look between $17.8ns$ and $18.0ns$ is given in Figure 2.21. We can see the magnitude of the impulse responses dies down as the time increase and m is found to be 13935. The hierarchical current constraints for

this example is shown in Table 2.3 where we use mA as the unit of current.

Table 2.3: The hierarchical current constraints.

Notation	Constraint
$L(\{1\}, 1.0), \ldots, L(\{5\}, 1.0)$	$I_1 \leq 1.0, \ldots, I_5 \leq 1.0$
$L(\{1, 2, 3\}, 2.0)$	$I_1 + I_2 + I_3 \leq 2.0$
$L(\{4, 5\}, 1.5)$	$I_4 + I_5 \leq 1.5$
$L(\{1, 2, 3, 4, 5\}, 2.5)$	$I_1 + I_2 + I_3 + I_4 + I_5 \leq 2.5$

We denote the slope constraint C by the transition time t_r where C equals the feasible upper bound of HCC in Table 2.3 divided by t_r. Then our algorithm is applied to generate the worst-case noise for two test cases with $t_r = 5ps$ and $t_r = 20ps$ respectively. The summation of the worst-case current stimuli corresponding to the time range in Figure 2.21 is shown in Figure 2.22 where we can see the feasible upper bound $2.5mA$ is almost unreachable when $t_r = 20ps$. It proves that assuming zero transition time will give unrealistic worst-case noise prediction. Figure 2.23 plots the transient voltage noise waveform of the power grid with the worst-case current stimuli when $t_r = 5ps$. The peak worst-case noise is at $t = T$.

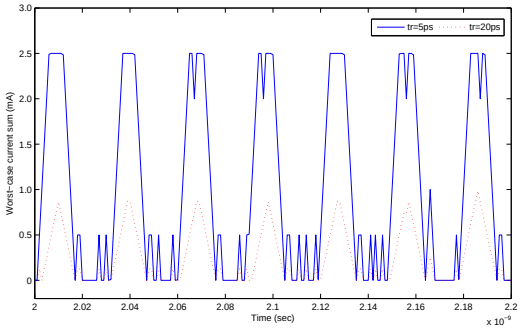


Figure 2.22: The worst-case currents with two different transition times.

Finally the run time of the algorithm for different pairs of n and s is given in Figure 2.24. It shows that the number of inputs is not the bottleneck of our algorithm because of theorem 4 and greedy evaluation of weight function. As a result, our algorithm works efficiently for an n which is large enough for on-chip power grid verification.

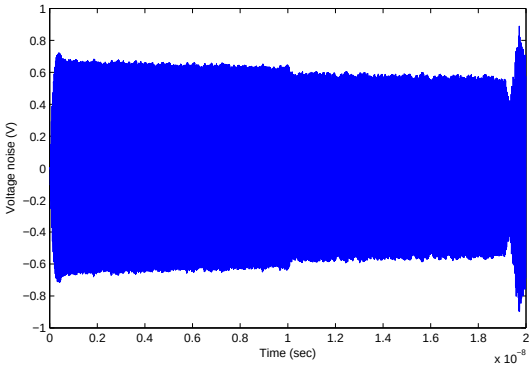


Figure 2.23: The voltage noise waveform for $t_r = 5ps$.

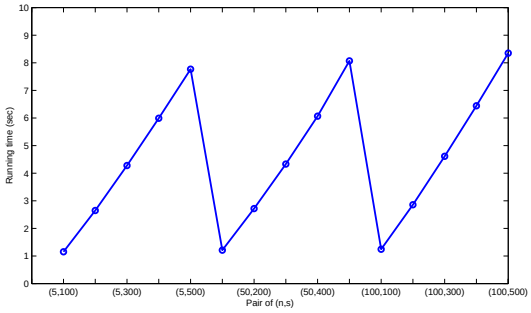


Figure 2.24: The comparison of running time for different pairs of (n, s).

Chapter 2, in part, is a reprint of the material as it appears in International Symposium on Quality of Electronic Design(ISQED) 2010. Peng Du; Xiang Hu; Shih-Hung Weng; Amirali Arani; Xiaoming Chen; A. Ege Engin; Chung-Kuan Cheng, IEEE, 2010. The dissertation author was the primary investigator and author of this paper.

Chapter 3

Power Grid Sizing

Since power integrity has become a critical issue in the VLSI design, generating a robust on-chip power/ground network to fulfill the design requirements has become a challenging task. As the manufacture technology scaling down, the growing transistor density and high clock frequency increase the supply current requirement, and cause large IR drop on the power/ground network. Although many advanced packaging technologies, e.g. flip-chip and C4 bump, have been applied for improving the quality of power delivery, the pushing of the supply voltage for low power designs reduces the noise margin.

Most previous works require a given average current density or peak magnitude as the constraint for each individual current source on a power grid to formulate an optimization problem. The average current density and peak magnitude rely on the logic simulation of a set of vector patterns. Although the logic simulation is fast, it is unlikely to cover all cases by exhaustive simulation. The unexpected vectors might induce IR drop violation in an optimized power grid design.

By contrast, the total current density or peak magnitude of a chip is already given as a design specification at the beginning. In this work, we relax the optimization problem by only considering the total current density/magnitude. By doing so, we can save the effort of logic simulation and are able to perform early stage power grid design verification before placement and routing. We devise a sizing method to minimize the worst possible IR drop on a fixed power grid topology.

We use the concept of effective resistance on a power grid to reduce the sizing optimization problem into a min-max convex programming problem. Experimental results show that our method can achieve 40% improvement for regular 2D grids and 7.32% improvement for a four layers power grid with only the top two layers tunable over uniform sizing.

3.1 Problem Formulation

Let $N(V, E)$ be an electrical network where $V = \{v_1, v_2, \ldots, v_n\}$ and $E = \{e_1, e_2, \ldots, e_m\}$ denote nodes and edges of the network respectively. For each edge $e \in E$, we use $g(e)$ to denote the conductance of edge e where g is a function from E to $\mathbb{R}^+$. In our power grid model, we assume currents flow into the network at a single node u and flow out at a set of nodes $W \subseteq V$. We use a function $I : W \rightarrow \mathbb{R}^+$ to denote a current source distribution on node set W with normalization constraint $\sum_{v \in W} I(w) = 1$. Let $D(v, g, I)$ be the voltage drop between u and v for given conductance assignment g and current profile I. Our objective is to minimize the worst voltage drop of the power grid over all locations v and current distributions I by allocating a fixed total conductance among the edges. Without loss of generality, we can assume that the total conductance to be allocated is one. Formally, our problem can be formulated as:

$$\begin{aligned} \min_{g:E\rightarrow\mathbb{R}^+} \quad & \max_{I:W\rightarrow\mathbb{R}^+} \max_{v\in V} D(v, g, I) \\ \text{s.t.} \quad & \sum_{e\in E} g(e) = 1 \end{aligned} \tag{3.1}$$

The original form of the objective function of problem (3.1) is non-differentiable and hard to predict directly. Hence we can not expect general non-linear programming methods to solve problem (3.1) efficiently. In Section 3.2, we simplify the prediction of worst voltage drop by computing the largest effective resistance of the network and prove problem (3.1) can be reduced into a convex programming problem.

3.2 Voltage Drops versus Effective Resistances

In this section, we first deduce an effective resistance representation of the objective function of problem (3.1). Then we will solve the problem of minimizing the largest effective resistance of the network, which is equivalent to (3.1), by convex programming techniques.

First we define $I_w : W \to \mathbb{R}^+$ as the current profile corresponding to a single current source at node w, i.e. $I_w(w) = 1$ and $I_w(v) = 0$ when $v \neq w$. The objective function of (3.1) can be simplified as:

$$\begin{aligned} & \max_{I:W\to\mathbb{R}^+} \max_{v\in V} D(v,g,I) \\ = & \max_{v\in V} \max_{I:W\to\mathbb{R}^+} D(v,g,I) \\ = & \max_{v\in V} \max_{I_w:w\in W} D(v,g,I_w) \\ = & \max_{w\in W} D(w,g,I_w) \\ = & \max_{w\in W} R_{uw}, \end{aligned} \tag{3.2}$$

where R_{uw} denotes the effective resistance between nodes u and w. The second equality can be proven by superposition since for a fixed node v, the voltage drop at it can be separated by contributions of current sources at different positions. Then there must be a position $w \in W$ who contributes most when unit current goes through it and we can pick I_w to generate worst voltage drop at v. The reason for the third equality is the fact that $D(v,g,I_w) \leq D(w,g,I_w)$. Now we have the following optimization problem equivalent to (3.1):

$$\begin{aligned} \min_{g:E\to\mathbb{R}^+} \quad & \max_{w\in W} R_{uw} \\ \text{s.t.} \quad & \sum_{e\in E} g(e) = 1 \end{aligned} \tag{3.3}$$

The constraint of the optimization problem (3.3) is obviously convex since it is a linear function on g. Therefore, to show (3.3) is a convex programming problem, we only need to prove R_{uw} is a convex function on g since the maximum of finite convex functions is still convex. Let $G = A \cdot D_g \cdot A^T$ be the weighted Laplacian matrix of the network N where D_g is the diagonal matrix formed by $g(e_1), g(e_2), \ldots, g(e_m)$ and A is the incidence matrix of N. Since $g(e) \geq 0$ for all $e \in$

E, G is positive semidefinite. By KCL,KVL and Ohm's law (see [GBS06],[S.B06]), we can deduce that

$$R_{uw} = (e_u - e_w)^T (G + \vec{1}\vec{1}^T/n)^{-1}(e_u - e_w), \tag{3.4}$$

where $\vec{1} \in \mathbb{R}^n$ is the vector with one everywhere and $e_w \in \mathbb{R}^n$ denotes the i^{th} unit vector if $w = v_i$. Since elements of G is linear on g and $G + \vec{1}\vec{1}^T/n$ is semidefinite, we conclude R_{uw} is a convex function on g (see [BV04b],§3.1.7).

3.3 Semidefinite Programming Reduction

If the scale of the power grid is relatively small (e.g. hundreds of nodes and edges), we can reformulate the problem (3.3) into a semidefinite programming problem as follows:

$$\begin{aligned} \min_{g:E\to\mathbb{R}^+} \quad & \max_{\{i:v_i\in W\}} (E^T Y E)_{ii} \\ \text{s.t.} \quad & \sum_{e\in E} g(e) = 1 \\ & \begin{bmatrix} G + \vec{1}\vec{1}^T/n & I \\ I & Y \end{bmatrix} \succeq 0, \end{aligned} \tag{3.5}$$

where $E \in M_{n\times n}(\mathbb{R})$ is the matrix whose i^{th} column equals to $e_u - e_{v_i}$ and $Y \in M_{n\times n}(\mathbb{R})$ is a relaxation matrix. When the optimal solution of problem (3.5) is achieved, we have $Y = (G + \vec{1}\vec{1}^T/n)^{-1}$ by the theory of Schur complement (see [BV04b],§A.5.5). Hence, problem (3.5) is a semidefinite programming problem (SDP) equivalent to problem (3.3). We can solve problem (3.5) by standard SDP solvers such as CVX [GB11].

3.4 Optimal Sizing by Convex Programming

In order to solve the sizing problem with large scalability, we first rewrite problem (3.3) as below:

$$\begin{aligned} \min_{g:E\to\mathbb{R}^+} \quad & \lambda \\ \text{s.t.} \quad & \sum_{e\in E} g(e) = 1 \\ & \lambda \geq R_{uw},\ \forall w \in W \end{aligned} \tag{3.6}$$

Since the objective function and constrains of (3.6) are smooth, we can adopt interior point method (see [BV04b],§11) or gradient descent method to solve it. The main obstacle on efficiency of solving problem (3.6) is the time complexity of evaluating the effective resistance and the derivative of effective resistance relative to perturbation of wire widths. We will introduce a Krylov space method which can compute the effective resistances between node u to all other nodes at the same time in section 3.4.1. If the scale of the power grid is large, we evaluate the derivative of effective resistance numerically by comparing effective resistances for two adjacent conductances distribution. Otherwise, we introduce an efficient method to gradually compute and update the derivative of effective resistance in section 3.4.2.

3.4.1 Lanczos Algorithm for Objective Evaluation

In this section, we will adopt a modified Lanczos algorithm first introduced in [Car10] to evaluate the effective resistance efficiently. First we define $\tilde{G}$ to be the matrix obtained by deleting the row and column corresponding to node u in the matrix G. Then the effective resistance R_{uw} is the diagonal element of $\tilde{G}^{-1}$ corresponding to the node w. Hence evaluating the effective resistance R_{uw} for all $w \in W$ can be reduced into computing the diagonal elements of the inverse of $\tilde{G}$. By computing the Krylov space of matrix $\tilde{G}$ up to the iteration m, we have the following equation:

$$\tilde{G}Q_m = Q_m T_m + \beta_{m+1} q_{m+1} e_m^T, \tag{3.7}$$

where T_m has the form

$$T_m = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{m-1} & \alpha_{m-1} & \beta_m \\ & & & \beta_m & \alpha_m \end{bmatrix} \tag{3.8}$$

and Q_m is a unitary matrix, and e_m^T is the unit vector with one at the m^{th} position. By the idea of the standard Lanczos algorithm, we can approximate the matrix $\tilde{G}$ as

$$\tilde{G} \approx Q_m T_m Q_m^T \tag{3.9}$$

Therefore, the calculation of the diagonal of the inverse of $\tilde{G}$ can be approximated using the same expression:

$$diag(\tilde{G}^{-1}) \approx diag(Q_m T_m^{-1} Q_m^T) \tag{3.10}$$

In order to evaluate the right-hand side of equation (3.10) efficiently, we decompose the matrix T_m as:

$$T_m = L_m D_m L_m^T, \tag{3.11}$$

where L_m is a matrix with the form:

$$L_m = \begin{bmatrix} 1 & 0 & & & \\ c_1 & 1 & 0 & & \\ & \ddots & \ddots & \ddots & \\ & & c_{m-2} & 1 & 0 \\ & & & c_m & 1 \end{bmatrix} \tag{3.12}$$

and D_m is a diagonal matrix. The values of $c_1, c_2, \ldots, c_{m-1}$ and the diagonal elements of D_m can be directly computed by the equation (3.11). Now we can rewrite the diagonal terms of the inverse of $\tilde{G}$ as:

$$\begin{aligned} diag(\tilde{G}^{-1}) &\approx diag((Q_m L_m D_m L_m^T Q_m^T)^{-1}) \\ &= diag(Q_m (L_m^T)^{-1} D_m^{-1} L_m^{-1} Q_m^T) \\ &= diag(P_m D_m^{-1} P_m^T), \end{aligned} \tag{3.13}$$

where $P_m = Q_m(L_m^T)^{-1}$. Since the matrix P_m can be evaluated iteratively from the process of the Lanczos algorithm [Car10] and D_m^{-1} is a diagonal matrix, we can efficiently obtain all diagonal elements of $\tilde{G}^{-1}$ by computing the right-hand side of equation (3.13).

3.4.2 Derivative of Effective Resistances

In this section, we will deduce a simple formula to evaluate the sensitivity of effective resistance relative to perturbation of wire widths. Without loss of generality, we assume $u = v_n$ and the matrix $\tilde{G}$ is equal to the matrix G with the last row and column deleted. Let B be the matrix equal to A with the last row deleted and we have $\tilde{G} = B \cdot D_g \cdot B^T$. By the similar argument to obtain the formula (3.4), we can deduce that

$$R_{uw} = e_w^T \cdot \tilde{G}^{-1} \cdot e_w \tag{3.14}$$

Let $E_{i,\epsilon}$ be the $n \times n$ matrix with ϵ being the i^{th} diagonal term of E and zero elsewhere. The derivative of R_{uw} relative to an edge conductance $g(e_i)$ can be computed as:

$$\begin{aligned} & \frac{\partial R_{uw}}{\partial g(e_i)} \\ = & \lim_{\epsilon \to 0} \frac{e_w^T((B(D_g + E_{i,\epsilon})B^T)^{-1} - \tilde{G}^{-1})e_w}{\epsilon} \end{aligned}$$

By the Sherman-Morrison formula, we can deduce that

$$\begin{aligned} & (B(D_g + E_{i,\epsilon})B^T)^{-1} \\ = & (\tilde{G} + BE_{i,\epsilon}B^T)^{-1} \\ = & (\tilde{G} + \epsilon\vec{v_i}\vec{v_i}^T)^{-1} \\ = & \tilde{G}^{-1} - \frac{\epsilon(\tilde{G}^{-1}\vec{v_i})(\tilde{G}^{-1}\vec{v_i})^T}{1 + \epsilon\vec{v_i}^T\tilde{G}^{-1}\vec{v_i}}, \end{aligned}$$

where $\vec{v_i}$ denotes the i^{th} column vector of B. Therefore, the derivative of R_{uw} relative to $g(e_i)$ can be simplified as:

$$\begin{aligned} & \frac{\partial R_{uw}}{\partial g(e_i)} \\ = & \lim_{\epsilon \to 0} \frac{-e_w^T(\tilde{G}^{-1}\vec{v_i})(\tilde{G}^{-1}\vec{v_i})^T e_w}{1+\epsilon \vec{v_i}^T \tilde{G}^{-1}\vec{v_i}} \\ = & -(e_w^T \tilde{G}^{-1}\vec{v_i})(e_w^T \tilde{G}^{-1}\vec{v_i})^T \\ = & -(e_w^T \tilde{G}^{-1}\vec{v_i})^2 \end{aligned} \tag{3.15}$$

In the process of optimization, we can update $\tilde{G}^{-1}$ gradually by Woodbury matrix identity. Suppose k conductances $g(e_{i_1}), g(e_{i_2}), \ldots, g(e_{i_k})$ have been changed in an iteration of optimization, we define $\tilde{B}$ as the matrix B with columns $i_1, i_2, \ldots, i_k$ deleted and $\tilde{D}_g$ as the diagonal matrix formed by $g(e_{i_1}), g(e_{i_2}), \ldots, g(e_{i_k})$. Let $\tilde{G}_o, \tilde{G}_n$ be the matrix $\tilde{G}$ corresponding to the original and updated conductances respectively. We can evaluate the inverse of $\tilde{G}_n$ from the inverse of $\tilde{G}_o$ as:

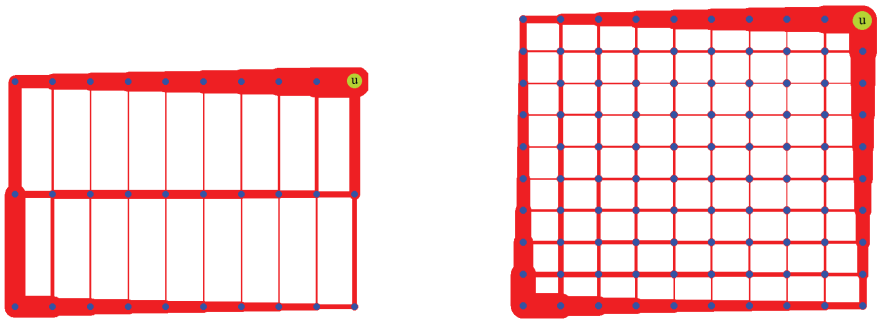
$$\begin{aligned} & \tilde{G}_n^{-1} \\ = & (\tilde{G}_o + \tilde{B}\tilde{D}_g\tilde{B}^T)^{-1} \\ = & \tilde{G}_o^{-1} - \tilde{G}_o^{-1}\tilde{B}R^{-1}\tilde{B}^T\tilde{G}_o^{-1}, \end{aligned} \tag{3.16}$$

where $R = \tilde{D}_g^{-1} + \tilde{B}^T\tilde{G}_o^{-1}\tilde{B}$. Hence, the evaluation of formula (3.16) involves computing the inverse of a $k \times k$ matrix R and multiplication of matrices whose total time complexity is $O(k^3)+O(n^2k)$ rather than $O(n^3)$ by evaluating the inverse of $\tilde{G}_n^{-1}$ directly.

3.5 Experimental Results on Regular Grids

We assume a regular 2D grid has R rows and C columns, and u locates at the top-right corner of the grid. We adopt the semidefinite programming formulation (3.5) and the package CVX [GB11] to find the optimal solution. For the largest case with 10 rows and 10 columns, the CVX solver takes 15 minutes and 82 iterations to converge. Figure 3.1(a) and 3.1(b) gives the optimal conductances distribution for 3×10 and 10×10 grids respectively. The blue nodes in these figures indicate

the wire segment endpoints and the width of each segment is proportional to the conductance assigned to it. We can observe that most conductance resource has been assigned to the top-right corner and bottom-left corner since the maximum effective resistance is generally obtained from u to the bottom-left node. Figure 3.2 gives the effective resistance map from the top-right node to all other nodes. The blue and green surfaces show the effective resistances for uniform and optimized wire widths respectively. We observe that both surfaces achieve the largest effective resistance at the bottom-left node and the green surface is below the blue surface on every node. The green surface is totally flat which reflects the optimality of our solution.



(a) Optimal wire widths (3×10). (b) Optimal wire widths (10×10).

Figure 3.1: Optimization result for regular 2D power grids.

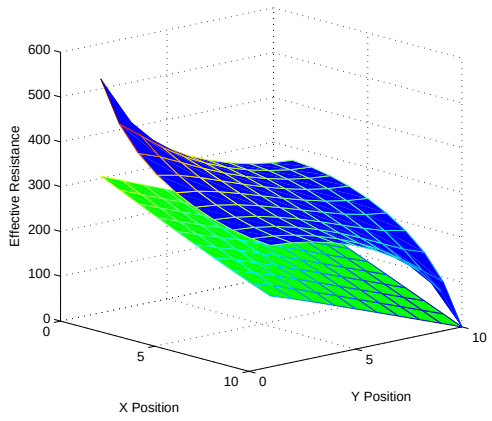


Figure 3.2: Effective resistances for a regular 10×10 grid.

Table 3.1 compares the maximum effective resistance (i.e. $\max_{w \in W} R_{uw}$) for uniform conductances distribution (column "MaxR(uni)") and for optimal conductances distribution (column "MaxR(opt)"). The column 'improvement' indicates the improvement on maximum effective resistance of the optimal solution relative to uniform conductances distribution. We can see that for large R, C values, our optimized sizing results can achieve up to 40% improvement over uniform sizing.

Table 3.1: Comparison for the maximum effective resistance

(R,C)	MaxR(uni)	MaxR(opt)	improvement
(3,3)	18.00	16.00	11.11%
(4,4)	44.57	36.00	19.23%
(4,6)	85.95	64.00	25.54%
(6,4)	85.95	64.00	25.54%
(5,5)	85.45	64.00	25.11%
(5,7)	142.21	100.00	29.68%
(7,5)	142.21	100.00	29.68%
(3,10)	174.19	121.00	30.53%
(10,3)	174.19	121.00	30.53%
(6,10)	304.71	196.00	35.68%
(10,10)	542.10	324.89	40.07%

3.6 Experimental Results on Practical Networks

Figure 3.3 shows a practical power grid model with four metal layers which are denoted by $M1$,$M3$,$M6$ and AP. The whole die size of this power grid is $200um \times 200um$. The network parameters including directions, pitches and initial wire sizes of each layer are shown in Table 3.2. For each layer, wires are placed in the corresponding direction with distance "Pitch" between two adjacent wires. The wires in adjacent layers are connected by ideal vias with zero resistance on intersection points. The initial wire widths for each layer is indicated in the column "Initial Width" of Table 3.2, which is proportional to the initial conductance of wires. In this setup, the number of wires for layers $M1$-AP are $81, 26, 11, 6$ respectively which is computed by $(200/Pitch + 1)$. The total number of nodes for layers

$M1$-AP are $2106(81\times26), 2392(81\times26+26\times11), 352(26\times11+11\times6), 66(11\times6)$ respectively.

Table 3.2: The network parameters of a power grid

Layer	Direction	Pitch	Initial Width	Width Range
M1	Horizontal	2.5um	0.17um	N/A
M3	Vertical	8.0um	0.25um	N/A
M6	Horizontal	20um	4.2um	2um-8um
AP	Vertical	40um	10um	3um-16um

For our power grid model, the voltage source (i.e. the node u) locates at the top-right corner of layer AP. The current sources are distributed at layer $M1$, i.e. the set W contains all nodes in $M1$. We can only tune wire widths of M6 and AP with width range constraints indicated in Column "Width Range" of Table 3.2 in order to minimize the worst voltage drop.

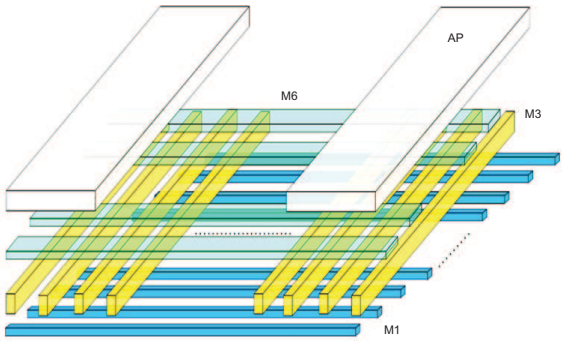


Figure 3.3: A power grid model with four layers.

We program a convex programming solver in Matlab using gradient descent method and run it on a machine with Intel Core i3 2.40GHz CPU and 4GB memory. It takes 26 minutes and 70 iterations for our solver to converge to the optimal solution within 0.1% relative error for the practical power grid. The peak memory usage when running the solver is 321MB. Figure 3.4 gives the optimal sizing result for our power grid model where red lines and green lines indicate the wire widths

of AP and M6 respectively. We can see that the middle wires become narrower and the outside wires become wider for both layers after optimization. Figure 3.5 shows the voltage map on layer M1 of the sizing result corresponding to the current distribution which generates the worst voltage drop, i.e. all current sources gather at the origin in M1. The worst voltage drop for the optimized sizing result is $0.2830V$ which is 7.32% improvement relative to the worst voltage drop $0.3054V$ for the uniform sizing. Since the worst voltage drop is achieved on bottom-left corner as opposed to the top-right voltage source, the whole width distribution is symmetric. Figure 3.6 gives two more sizing results when the width range of M6 and AP changes respectively and Table 3.3 lists the worst voltage drops for different width range alternation. Figure 3.7 shows the worst voltage drops when the area resource is transformed between layer M6 to AP. The "Adjust Percentage" indicates the ratio of adjustment on M6. The curve is monotonically decreasing which means that assigning more resource to M6 gives smaller worst voltage drop.

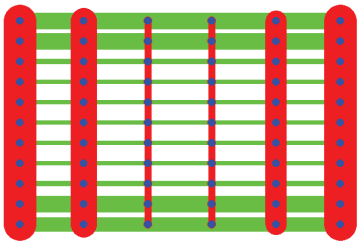


Figure 3.4: Sizing result for the practical power grid.

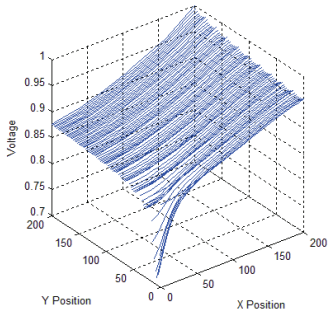
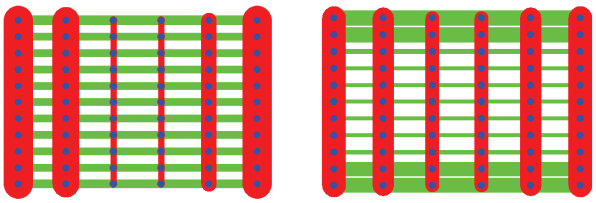


Figure 3.5: Voltage map of layer M1.



(a) M6 width range = 4um-6um.

(b) AP width range = 7um-12um.

Figure 3.6: Sizing result when width range changes.

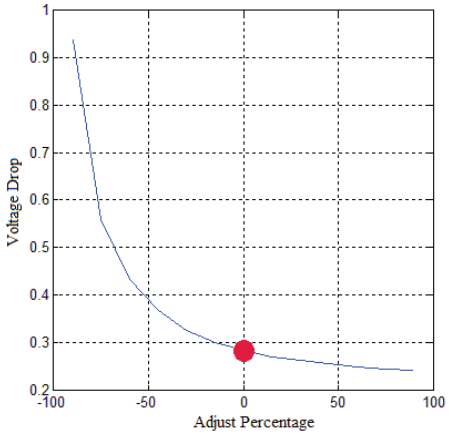


Figure 3.7: Voltage drops for different area distributions.

Table 3.3: Comparison for different width ranges

Width Range (M6)	Width Range (AP)	Voltage Drop
2um-8um	3um-16um	0.2830V
3um-7um	3um-16um	0.2855V
4um-6um	3um-16um	0.2891V
2um-8um	5um-14um	0.2858V
2um-8um	7um-12um	0.2888V
3um-7um	5um-14um	0.2885V
3um-7um	7um-12um	0.2920V
4um-6um	5um-14um	0.2927V
4um-6um	7um-12um	0.2967V

Chapter 3, in full, is a reprint of the material as it appears in IEEE International Conference on ASIC 2011. Peng Du; Shih-Hung Weng; Xiang Hu; Chung-Kuan Cheng, IEEE, 2011. The dissertation author was the primary investigator and author of this paper.

Chapter 4

Optimization for dynamic power distribution network

In this chapter, we propose a whole flow for early-stage verification and optimization of dynamic power distribution network. The verification part generates the worst current profile satisfying duty-cycle constraints, group magnitude constraints and transition time constraints which makes the prediction more realistic. The optimization part optimizes the worst violation area by allocating decoupling capacitors (Decap) and controlled equivalent series resistors (ESR). We run the verification and optimization flow alternatively in order to find the best network parameters which minimizes the worst violation area. An adjoint network technique is adopted to evaluate the sensitivity of the worst violation area relative to current sources and network parameters. Then the sequential quadratic programming method is adopted for solving the proposed non-linear programming problem. In order to simulate the original network and adjoint network for large scale circuits, we introduce two methods including adaptive DFT and matrix exponential which are much faster than SPICE library.

4.1 Problem Formulation

We consider the power grid noise as the violation area at a circuit node i, which is defined as:

$$g_i = \int_0^T (V_{min} - v_i(t))\chi(v_i(t) < V_{min})dt, \tag{4.1}$$

where V_{min} is a prescribed voltage lower bound, $v_i(t)$ is the voltage curve of node i and χ is the indicator function. Our objective is to minimize the worst case of the sum of violation areas over N output nodes by tuning Decap and ESR. The worst case is evaluated subject to three type of constraints assumed for load currents described as follows. Suppose there are n current sources denoted by $I_1(t), I_2(t), \ldots, I_n(t)$ where $t \in [0, T]$. One class of constraints concerning the burden of circuits with respect to time intervals $[t_0 = 0, t_1], [t_2, t_3], \ldots, [t_{m-1}, t_m = T]$ is formulated as:

$$\begin{aligned}
&I_k(t) \geq 0, \;\; 1 \leq k \leq n \\
&I_1(t) \leq B_{1,1}, \ldots, I_n(t) \leq B_{1,n}, t \in [t_0, t_1] \\
&I_1(t) \leq B_{2,1}, \ldots, I_n(t) \leq B_{2,n}, t \in [t_1, t_2] \\
&\ldots\ldots \\
&I_1(t) \leq B_{m,1}, \ldots, I_n(t) \leq B_{m,n}, t \in [t_{m-1}, t_m],
\end{aligned} \tag{4.2}$$

where $B_{i,j}$ is the upper bound of $I_j(t)$ on time interval $[t_{i-1}, t_i]$. In order to take into account the correlation between current sources, we assume there are p groups $G_1, G_2, \ldots, G_p \subseteq \{1, \ldots, n\}$ such that the sum of currents in G_i is bounded above by a constant D_i, i.e.

$$\sum_{k \in G_i} I_k(t) \leq D_i, \;\; 1 \leq i \leq p \tag{4.3}$$

Since the transition time of load currents is non-zero in practice, we add other constraints for the slope of load currents as follows,

$$-L_k \leq \frac{dI_k(t)}{dt} \leq L_k, \;\; 1 \leq k \leq n, \tag{4.4}$$

where L_k is a constant upper bound.

We denote the amount of Decap and ESR at each candidate location $i \in [1, M]$ by C_i and R_i respectively where M is the number of candidate locations. For the power grid optimization, we choose C_i and R_i for every location so that the worst voltage violation area with respect to three kinds of current constraints discussed above is minimized. There are also budge constraint on total Decap and space constraint on each location which is shown below:

$$\begin{aligned} &\sum_{i=1}^{M} C_i \leq Q, \ \text{(Decap budget constraint)} \\ &0 \leq C_i \leq M_i^C, \text{ (Space constraint on Decap)} \\ &0 \leq R_i \leq M_i^R, \text{ (Space constraint on ESR),} \end{aligned} \tag{4.5}$$

where M_i^C and M_i^R for $1 \leq i \leq M$ are constant upper bound for Decap and ESR on location i, respectively.

Let $\vec{I}(t) = (I_1(t), \ldots, I_n(t))$, $\vec{C} = (C_1, \ldots, C_M)$ and $\vec{R} = (R_1, \ldots, R_M)$. We define $g_i(\vec{I}(t), \vec{C}, \vec{R})$ as the voltage violation area on node i with respect to these three vectors. Therefore, the evaluation of worst case violation area when Decap and ESR are fixed can be formulated as following optimization problem:

$$\begin{aligned} \max_{\vec{I}(t)} \quad & \sum_{i=1}^{N} g_i(\vec{I}(t), \vec{C}, \vec{R}) \\ \text{s.t.} \quad & \text{Duty Cycle Constraints (4.2),} \\ & \text{Group Magnitude Constraints (4.3),} \\ & \text{Transition Time Constraints (4.4).} \end{aligned} \tag{4.6}$$

We use $g^w(\vec{C}, \vec{R}) = \max_{\vec{I}(t)} \sum_{i=1}^{n} g_i(\vec{I}(t), \vec{C}, \vec{R})$ to denote the worst case noise obtained from previous optimization problem and our final objective can be formulated as:

$$\begin{aligned} \min_{\vec{C}, \vec{R}} \quad & g^w(\vec{C}, \vec{R}) \\ \text{s.t.} \quad & \sum_{i=1}^{M} C_i \leq Q, \\ & 0 \leq C_i \leq M_i^C, \\ & 0 \leq R_i \leq M_i^R. \end{aligned} \tag{4.7}$$

4.2 The Overall Flow

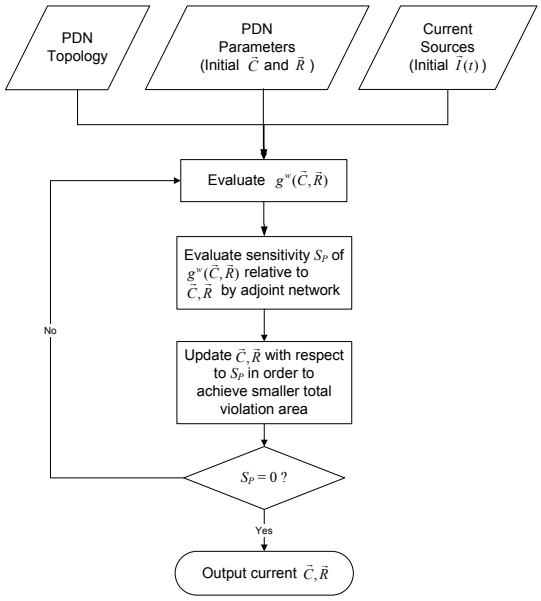


Figure 4.1: The overall flow of worst noise optimization.

The whole flow of our worst noise optimization is shown in Figure 4.1. The input of the optimization procedure is a PDN netlist which includes PDN topology, initial Decaps $\vec{C}$, ESRs $\vec{R}$ and current sources $\vec{I}(t)$. We first evaluate the worst total violation area $g^w(\vec{C},\vec{R})$ with current sources satisfying constraints (4.2),(4.3),(4.4) and $\vec{C},\vec{R}$ fixed. The detail of computing $g^w(\vec{C},\vec{R})$ is shown in Figure 4.2. Secondly, we fixed the current sources achieving $g^w(\vec{C},\vec{R})$ and evaluate the sensitivity of $g^w(\vec{C},\vec{R})$ relative to $\vec{C},\vec{R}$ by adjoint network. Finally, we update $\vec{C},\vec{R}$ corresponding to the sensitivity under constraints (4.5) and finish the optimization process until zero sensitivity is obtained. In practice, we adopt sequential quadratic programming method to find best possible $\vec{C}$ and $\vec{R}$.

Figure 4.2 describes the flow of evaluating $g^w(\vec{C},\vec{R})$. At first, we construct a PDN network based on current $\vec{C},\vec{R}$ and obtain the total voltage violation area $G = \sum_{i=1}^{N} g_i(\vec{I}(t),\vec{C},\vec{R})$ by simulation. After that, we compute the sensitivity of G relative to $\vec{I}(t)$ and update $\vec{I}(t)$ correspondingly to achieve worse total violation area. The process continues until zero sensitivity is achieved and we return current

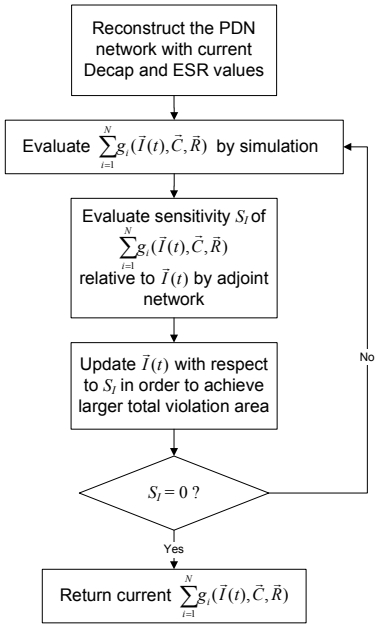


Figure 4.2: The flow of predicting worst noise.

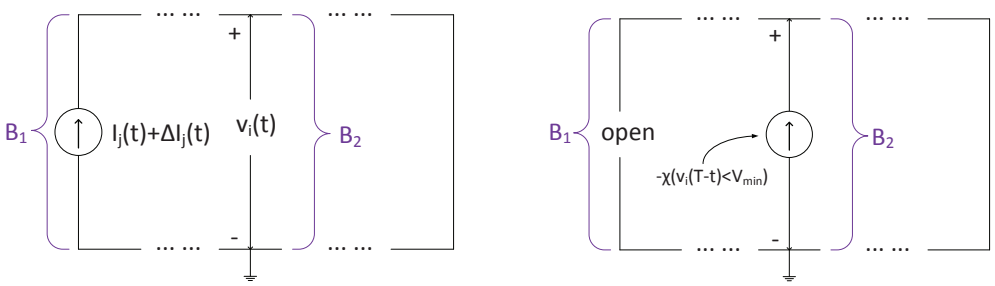
G as $g^w(\vec{C},\vec{R})$ which is the worst total voltage violation area.

4.3 Sensitivity Evaluation by Adjoint Network

In this section, we deduce the formulas for evaluating sensitivities of voltage violation area $g_i(\vec{I}(t),\vec{C},\vec{R})$ relative to $\vec{I}(t),\vec{C}$ and $\vec{R}$ respectively by adjoint network approach. We define S_B to be the set of all branches of the power grid. For each $B \in S_B$, we use $v^B(t), i^B(t)$ and $\tilde{v}^B(t), \tilde{i}^B(t)$ to denote the voltage and current of branch B in original network and adjoint network respectively.

- Sensitivity of $g_i(\vec{I}(t),\vec{C},\vec{R})$ with respect to $\vec{I}(t)$:

  As in Figure 4.3, our objective is to evaluate the sensitivity of violation area g_i of voltage $v_i(t)$ in branch B_2 relative to the current source $I_j(t)$ in branch B_1. We first add a variation $\Delta I_j(t)$ to the current source $I_j(t)$ which forms the original network in Figure 4.3(a). The adjoint network shown in Figure 4.3(b) is constructed by adding a current source $-\chi(v_i(T-t) < V_{min})$ in branch B_2 and leaving the branch B_1 open. Note that χ indicates the



(a) Original network. (b) Adjoint network.

Figure 4.3: Sensitivity calculation relative to $\vec{I}(t)$.

characteristic function. Now we have the following two formulas concerning the branch voltages and currents in branches B_1 and B_2.

$$\begin{aligned} & \int_0^T (\tilde{v}^{B_1}(T-t)i^{B_1}(t) - v^{B_1}(t)\tilde{i}^{B_1}(T-t))dt \\ = & \int_0^T \tilde{v}^{B_1}(T-t)(I_j(t) + \Delta I_j(t))dt \\ = & \int_0^T \tilde{v}^{B_1}(T-t)I_j(t)dt \\ + & \int_0^T \tilde{v}^{B_1}(T-t)\Delta I_j(t)dt \end{aligned} \tag{4.8}$$

$$\begin{aligned} & \int_0^T (\tilde{v}^{B_2}(T-t)i^{B_2}(t) - v^{B_2}(t)\tilde{i}^{B_2}(T-t))dt \\ = & \int_0^T v_i(t)\chi(v_i(t) < V_{min})dt \\ = & \int_0^T V_{min}\chi(v_i(t) < V_{min})dt - g_i \end{aligned} \tag{4.9}$$

By applying Tellegen's theorem, we have

$$\begin{aligned} \sum_{B\in S_B} \int_0^T & (\tilde{v}^B(T-t)i^B(t) \\ - & v^B(t)\tilde{i}^B(T-t))dt = 0. \end{aligned} \tag{4.10}$$

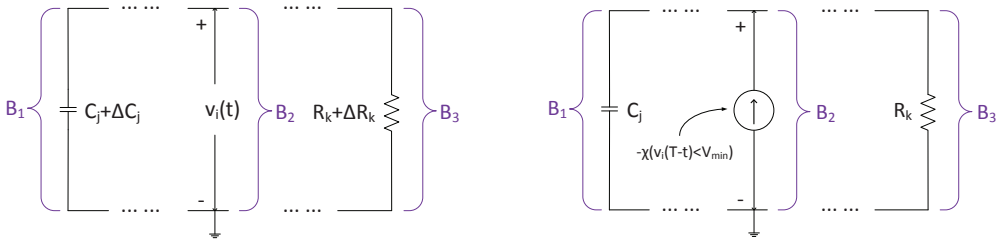
By plugging 4.8 and 4.9 into 4.10, we obtain

$$g_i = \int_0^T \tilde{v}^{B_1}(T-t)\Delta I_j(t)dt + M_1 \tag{4.11}$$

Therefore, the sensitivity of g_i relative to $I_j(t)$ can be derived as

$$\frac{\partial g_i}{\partial I_j(t)} = \tilde{v}^{B_1}(T-t), \qquad \forall t \in [0,T] \tag{4.12}$$

- Sensitivity of $g_i(\vec{I}(t), \vec{C}, \vec{R})$ with respect to $\vec{C}, \vec{R}$:



(a) Original network. (b) Adjoint network.

Figure 4.4: Sensitivity calculation relative to $\vec{C}, \vec{R}$.

In order to evaluate the sensitivities of violation area g_i of voltage $v_i(t)$ in branch B_2 relative to Decap C_j in branch B_1 and ESR R_k in branch B_3, we construct the original network and adjoint network as in Figure 4.4(a) and 4.4(b) respectively. By the similar argument as previous section, we have

$$\begin{aligned} g_i &= -\tilde{v}^{B_1}(T) C_j v^{B_1}(0) \\ &+ \int_0^T \tilde{v}^{B_1}(T-t)\dot{v}^{B_1}(t)\Delta C_j dt \\ &- \int_0^T i^{B_3}(t)\tilde{i}^{B_3}(T-t)\Delta R_k dt, \end{aligned} \tag{4.13}$$

and the sensitivities are derived as

$$\frac{\partial g_i}{\partial C_j} = \int_0^T \tilde{v}^{B_1}(T-t)\dot{v}^{B_1}(t)dt \tag{4.14}$$

$$\frac{\partial g_i}{\partial R_k} = -\int_0^T i^{B_3}(t)\tilde{i}^{B_3}(T-t)dt \tag{4.15}$$

4.4 Adaptive DFT Method for Circuit Simulation

In this section, the adaptive simulation method based on DFT is introduced. A PDN is usually extracted as a linear circuit model which contains resistors (R's), inductors (L's), capacitors (C's), and independent and dependent sources. The switching currents of load transistors are usually modeled as input current sources. The current profiles are extracted based on the switching activities of the load circuits and are usually given as a set of discrete values at uniform time points.

The basic DFT flow is the core of the adaptive simulation method. Figure 4.5 describes the basic DFT flow which computes the output voltage for the input with a frequency range of $[F_l, F_u]$ and a period of T. The input is the frequency-domain current which is converted from the time-domain input current by using DFT. A complex matrix is generated from the PDN linear circuit at each frequency sampling point. Then the complex matrix is solved at each frequency point to obtain the frequency-domain voltage response of the output. Finally, the time-domain output voltage is obtained by applying inverse DFT (IDFT) to its frequency-domain representation.

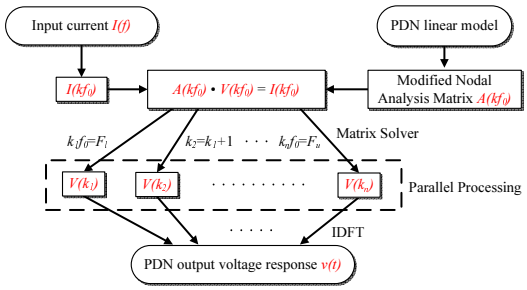


Figure 4.5: Compute output $v(t)$ in $[F_l, F_u]$ with period T, where $f_0 = 1/T$ and $F_l \leq kf_0 \leq F_u$.

Based on the wrapping effect of DFT, enough padding zeros must be added at the end of the original input in order to eliminate the wrap-around effect. The number of padding zeros can be very large depending on the system characteristics of the PDN. In addition, a small time interval has to be used to cover the whole

frequency range of the input current. This means that if we compute the output for the whole frequency range, the total number of sampling points to be solved will be large and the simulation time will be long.

Figure 4.6 illustrates the adaptive flow to address the problem above. The basic idea is to obtain the different parts of the output voltage by simulating the PDN with different periods and sampling intervals of the input. The tail of the PDN output voltage is always a low-frequency oscillation due to the "low-pass" nature of power distribution networks. This means that even though the period needs to be long, the tail can be captured accurately with a large sampling interval. For the main part of the output, a small sampling interval is needed to cover the whole frequency range of the input. However, the period can be short since the wrap-around effect can be canceled out by subtracting the captured tail from the output. With recursive partition, the total number of sampling points can be reduced significantly with little loss of accuracy.

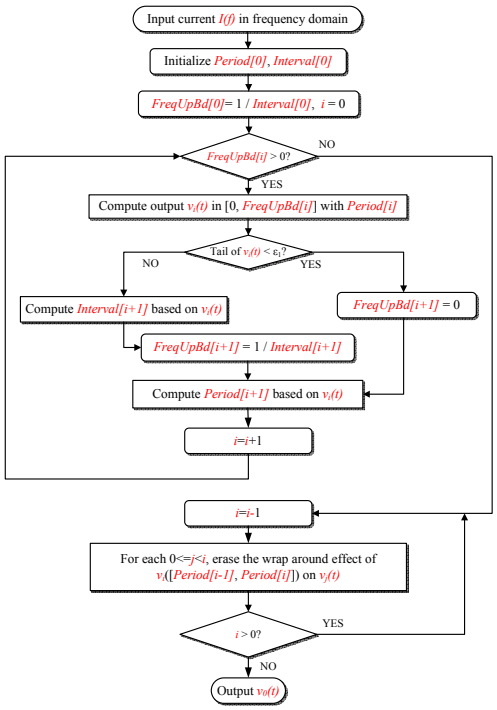


Figure 4.6: Adaptive DFT flow for PDN simulation.

4.4.1 Discrete Fourier Transform

Given an input current $i(t_n)$ with N discrete values at evenly spaced time points, i.e., $t_n = nt_0$ for $n = 0 \cdots N-1$, where t_0 is the time interval. Its discrete Fourier transform can be represented as

$$I(f_k) = \frac{1}{N} \sum_{n=0}^{N-1} i(t_n) e^{-j2\pi nk/N}, k = 0 \cdots N-1. \tag{4.16}$$

The input current is usually time limited, which means that beyond the duration of the input data the current is zero. The DFT equation implies that the input is extended to a periodic function $i_T(t_n)$ with a period T, where $T = Nt_0$, i.e., $i_T(t_n+NT) = i(t_n)$ for arbitrary integer n. The discrete Fourier transform of $i_T(t_n)$, $I_F(f_k)$, is also periodic with period F, where $F = 1/t_0 = N/T$. In Eqn.(4.16), $I(f_k)$ is the value of $I_F(f_k)$ within one period. It has N distinct values at evenly spaced frequency points f_k, i.e., $f_k = kf_0$ for $k = 0 \cdots N-1$, where $f_0 = 1/T = F/N$.

The input of the adaptive flow, i.e., $I(f)$, is the frequency-domain current, which is converted by using DFT from the original input current with an enough number of padding zeros, i.e., a long enough period T. This current is regarded as the reference input since its interval in both time-domain and frequency-domain is small enough. At each call of the "output computation" as shown in Figure 4.5, the frequency-domain information of the input can be simply obtained from the reference input based on the period T and sampling interval t_0.

Similarly, inverse discrete Fourier transform (IDFT) is used to convert the frequency-domain voltage response back to time domain, as represented in Eqn.(4.17):

$$v(t_n) = \sum_{k=0}^{N-1} V(f_k) e^{j2\pi nk/N}, n = 0, \ldots, N-1. \tag{4.17}$$

4.4.2 Matrix Solver in Frequency Domain

Given the circuit netlist extracted from the PDN model, a matrix is generated and a set of linear equations in frequency domain can be established as

$$A(f)\mathbf{x}(f) = \mathbf{b}(f), \tag{4.18}$$

where $A(f)$ is the generated complex matrix, $\mathbf{x}(f)$ is a vector containing the voltage at each node of the PDN, and $\mathbf{b}(f)$ is a vector which includes the input current sources. By solving these linear equations, frequency-domain nodal voltage responses of the PDN can be obtained.

In this section, modified nodal analysis (MNA) is used to generate the matrix. With modified nodal analysis, the matrix can be generated by a simple stamping process. For circuit elements with frequency dependent values, simply stamping the corresponding values to the matrix can obtain the MNA matrix at each frequency sample point f_k.

Various efficient and robust linear matrix solvers can be used to solve the equations. For the cases with small matrix size, direct matrix solvers such as KLU are good choices. However, the computation time of direct matrix solvers becomes unacceptably long for large matrix sizes. In those cases, linear iterative solvers can be applied, such as Generalized Minimal Residual (GMRES) and Conjugate Gradient (CG). To have faster convergence, preconditioners such as Incomplete LU (ILU) are applied before solving the matrix. Since the process of solving the matrix at each frequency point is totally independent of each other, parallel computation can be applied.

4.4.3 Adaptive Flow for Frequency Partition

In this subsection, the adaptive flow is described in details. As shown in Figure 4.6, The flow starts with initial "$Period[0]$", "$Interval[0]$" and "$FreqUpBd[0]$", where "$Period[i]$", "$Interval[i]$" and "$FreqUpBd[i]$" denote the period of the input, the sampling interval and the upper bound of the input frequency range at each iteration, respectively. Note that $Interval[i] = 1/FreqUpBd[i]$. At each iteration, an tentative time-domain output $v_i(t)$ is computed within the frequency range of [0,$FreqUpBd$]. The upper bound of the frequency range and the sampling interval for the next iteration, i.e., $FreqUpBd[i+1]$ and $Interval[i+1]$, are obtained by estimating the oscillation frequency of the tail of $v_i(t)$. The period for the next iteration, i.e., $Period[i+1]$ can also be obtained by estimating the damping rate of the oscillation at the end of $v_i(t)$. Based on $Interval[i+1]$ and $Period[i+1]$,

the flow goes to the next iteration and the lower-frequency part in the range of $[0, FreqUpBd[i+1]]$ is processed. The iteration ends when $FreqUpBd[i+1]$ is equal to $FreqUpBd[i]$ or zero. Each iteration captures the tail that oscillates with one resonant frequency of the PDN. Then starting from the last iteration, the tail obtained at each iteration is subtracted from the output obtained from the previous iterations. In this way, the wrap-around effect in the output of each iteration is eliminated. The final result is obtained by the combination of the output at the first iteration and the tails obtained at the following iterations.

4.4.4 Time Complexity

From the flow description above it can be seen that the period and sampling interval used for the output computation at each iteration is adjusted dynamically. Although the period needed to capture the lower-frequency tail may be longer, the sampling interval can be larger. Thus, we do not have to use the smallest interval and longest period for the calculation over entire frequency range. And the total simulation time can be reduced.

The simulation time of the flow is determined by the total number of sampling points solved. Let T_i and Δt_i denote the period and sampling interval at each iteration, respectively. The time complexity of the adaptive flow is

$$\text{Adaptive flow time complexity} = O(\sum_{i=0}^{k-1} (T_i/\Delta t_i)), \tag{4.19}$$

where k is the number of iterations. In Eqn.(4.19), $T_0 < T_1 < \cdots < T_{k-1}$, and $\Delta t_0 < \Delta t_1 < \cdots < \Delta t_{k-1}$. On the other hand, the time complexity of the non-adaptive DFT flow is

$$\text{Non-adaptive flow time complexity} = O(T_{k-1}/\Delta t_0)). \tag{4.20}$$

Usually even the highest resonant frequency of the PDN is in the range of $100MHz$, which is much smaller than the frequency range of the input. Thus, Δt_i, $i = 1, \ldots, k-1$, can be much larger than Δt_0, resulting in a significant reduction in the adaptive flow simulation time.

4.5 Matrix Exponential Method for Circuit Simulation

In this section, we propose an explicit numerical integration method by matrix exponential to perform the transient analysis without suffering the stability issue. Instead of using traditional polynomial approximation for the numerical integration, our method uses the numerical solution of the ordinary differential equations of the circuit to avoid the stability problem and keep the accuracy. We devise a matrix partition algorithm to reduce the computation time of matrix exponential and keep sparsity of the partitioned matrices. Our numerical integration method can still have the efficiency of explicit methods by the spare matrix-vector multiplication.

4.5.1 Circuit Status Equation

In this section, we present a numerical solution of ordinary differential equations of the circuit, and also show the benefits of this numerical solution. In general, we can express an electronic circuit using ordinary differential equations in time domain as follows:

$$\dot{\mathbf{C}}\mathbf{x}(t) = \mathbf{G}\mathbf{x}(t) + \mathbf{b}(t), \tag{4.21}$$

where the $\mathbf{x}(t)$ is the vector of nodal voltages and branch currents, $\mathbf{b}(t)$ is the input voltage and current sources, $\mathbf{G}$ is the matrix describing the resistances and incidence between voltages and currents, and $\mathbf{C}$ describes capacitances and inductances [TWKA06].

The exact solution of equation (4.21) can be derived as

$$\mathbf{x}(t) = e^{\mathbf{A}t} \int_{t_0}^{t} e^{-\mathbf{A}\tau}\mathbf{u}(\tau)d\tau + e^{\mathbf{A}(t-t_0)}\mathbf{x}(t_0), \tag{4.22}$$

where $\mathbf{A}$ is $\mathbf{C}^{-1}\mathbf{G}$, and $\mathbf{u}(t)$ is $\mathbf{C}^{-1}\mathbf{b}(t)$. Because the analytic form of the exact solution is difficult to obtain, we approximate the solution by only calculating $\mathbf{x}(t)$ for some discrete value of t. Therefore, in equation (4.22), we assume $t_0 = T$ and

$t = T + h$, then the equations can be rewritten as

$$\mathbf{x}(T+h) = e^{\mathbf{A}h}\mathbf{x}(T) + e^{\mathbf{A}(T+h)} \int_{T}^{T+h} e^{-\mathbf{A}\tau}\mathbf{u}(\tau)d\tau, \tag{4.23}$$

where h is the chosen time step size. We can assume the value of each input source is constant within every time step. Note that this assumption can be hold when h is small enough. Thus, equation (4.23) can be approximated by the following equation.

$$\mathbf{x}(T+h) = e^{\mathbf{A}h}\mathbf{x}(T) + (e^{\mathbf{A}h} - \mathbf{I})\mathbf{A}^{-1}\mathbf{u}(T) \tag{4.24}$$

Equation (4.24) shows a numerical solution of equation (4.21). By this equation, we can have the voltage of a node at each time instant in an explicit way. This numerical solution is A-stable because it will tend to a fixed point exponentially when the eigenvalue of $\mathbf{A}$ is less than zero, which is usually hold for electronic circuits. Intuitively, this approximation is more accurate than forward and backward Euler, because forward and backward Euler can be derived from approximating $e^{\mathbf{A}h}$ as $(\mathbf{I} + \mathbf{A}h)$ and $(\mathbf{I} - \mathbf{A}h)^{-1}$.

Figure 4.7 illustrates an example to demonstrate the benefits of equation (4.24). The configuration of the circuit is: $R_1 = 0.1\Omega$, $R_2 = 50\Omega$, $R_3 = R_4 = 0.03\Omega$, $L = 45p$H, $C = 1f$F , and $C_{gnd} = 1p$F. The input voltage ramps from 0V to 1V and the transition time is 1ns. We apply forward Euler, backward Euler and the numerical solution to perform transient analysis. Figure 4.8 shows the numerical solution and forward Euler where the time step sizes are both 5×10^{-13}. The numerical solution is stable while forward Euler method causes a divergence. Figure 4.9 shows a close look of the numerical solution and backward Euler. The time step size of the numerical solution is also 5×10^{-13}. We can see that the result of backward Euler method is almost the same as of the numerical solution only when step size is 1×10^{-14}. The numerical solution is more accurate than backward Euler.

Although the numerical solution is stable and accurate, there are two problems we have to overcome. First, computing exponential of a large scale matrix is difficult. Even we can derive the matrix exponential, it is usually a full matrix

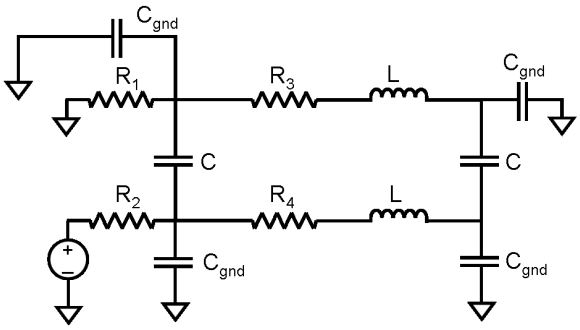


Figure 4.7: Example Circuit

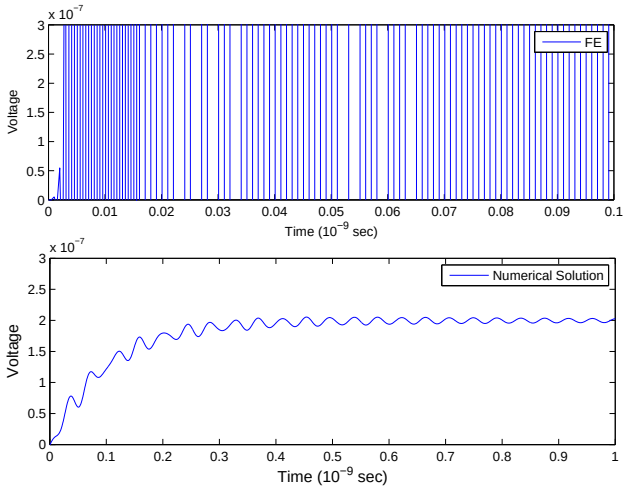


Figure 4.8: Numerical Solution and Forward Euler

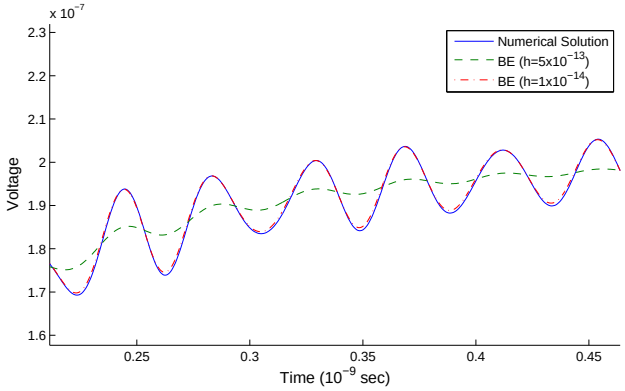


Figure 4.9: Numerical Solution and Backward Euler

which increases the complexity of matrix multiplication. However, the basic operation of explicit methods is the matrix multiplication. Therefore, the performance of explicit methods will significantly deteriorate with the matrix exponential.

Second, $\mathbf{C}^{-1}$ and $\mathbf{A}^{-1}$ are not easy to derive. Devgan and Rohrer [DR93] proposed an method to handle $\mathbf{C}^{-1}$ without doing the matrix inverse. The matrix $\mathbf{C}$ can be split into a diagonal matrix $\mathbf{\Lambda}$ and an off-diagonal matrix $\mathbf{N_n}$. Then, equation (4.21) can be rewritten as

$$\mathbf{\Lambda}\dot{\mathbf{x}}(T+h) = \mathbf{G}\mathbf{x}(T+h) + [\mathbf{b}(T+h) + \mathbf{N_n}\dot{\mathbf{x}}(T)].$$

The term $\dot{\mathbf{x}}(T)$ has already known when we calculate the value of $T+h$. Because $\mathbf{\Lambda}$ is a diagonal matrix, the inverse is just the reciprocal of its diagonal term. The matrix $\mathbf{A}$ and vector $\mathbf{u}(t)$ can be expressed as

$$\begin{aligned} \mathbf{A} &= \mathbf{\Lambda}^{-1}\mathbf{G} \\ \mathbf{u}(t) &= \mathbf{\Lambda}^{-1}[\mathbf{b}(T+h) + \mathbf{N_n}\dot{\mathbf{x}}(T)] \end{aligned}$$

Although this method can avoid computing $\mathbf{C}^{-1}$, finding $\mathbf{A}^{-1}$ is still the bottleneck in the numerical integration. We will present how we address these two problems in next section.

4.5.2 Fast Explicit Numerical Integration Method

In this section, we first present a high-order and efficient matrix exponential computation method by a matrix partition algorithm. With our method, the sparsity of the matrix can still be kept. Then, we show a fast numerical integration method by combining the matrix exponential method and an approximated numerical solution.

In order to evaluate $e^{\mathbf{A}h}$ where $\mathbf{A}$ is a sparse and large scale matrix, we first consider the definition of exponent of matrices as below:

$$e^{\mathbf{A}h} = I + \mathbf{A}h + \frac{(\mathbf{A}h)^2}{2!} + \frac{(\mathbf{A}h)^3}{3!} + \dots \tag{4.25}$$

Since $\mathbf{A}$ may contain millions of rows and columns, it is hard to compute the power of $\mathbf{A}$ efficiently in formula (4.25). The well-known Trotter decomposition

$$e^{(\mathbf{A_1}+\mathbf{A_2})h} = e^{\mathbf{A_1}h}e^{\mathbf{A_2}h} + O(h^2) \tag{4.26}$$

gives a possible way to partition the matrix $\mathbf{A} = \mathbf{A_1} + \mathbf{A_2}$ and approximates $e^{\mathbf{A}h}$ by the product of the exponential operators on the right-hand side in equation (4.26) with correction terms of the second order of h. Here we use a better approximation proposed in [HS05], whose correction terms are the fifth-order of h. At first, the symmetrization of Trotter formula is defined as follows:

$$S_2(h) \equiv e^{\frac{\mathbf{A_1}h}{2}} e^{\mathbf{A_2}h} e^{\frac{\mathbf{A_1}h}{2}} = e^{(\mathbf{A_1}+\mathbf{A_2})h + R_3 h^3 + R_5 h^5 + \cdots}, \tag{4.27}$$

where R_3 and R_5 denotes remainder terms with order 3 and 5 respectively. The symmetrized approximant has the property

$$S_2(h)S_2(-h) = e^{\frac{\mathbf{A_1}h}{2}} e^{\mathbf{A_2}h} e^{\frac{\mathbf{A_1}h}{2}} e^{\frac{-\mathbf{A_1}h}{2}} e^{-\mathbf{A_2}h} e^{\frac{-\mathbf{A_1}h}{2}} = I \tag{4.28}$$

which gives the reason of the vanishment of the even-order terms in the exponent of the right-hand side of (4.27).

Now, let s be a constant to be determined later, we define the fourth-order approximant for $e^{\mathbf{A}h}$ by

$$\begin{aligned} S(h) &\equiv S_2(sh)S_2((1-2s)h)S_2(sh) \\ &= e^{(\mathbf{A_1}+\mathbf{A_2})sh + R_3 s^3 h^3 + O(h^5)} \\ &\quad e^{(\mathbf{A_1}+\mathbf{A_2})(1-2s)h + R_3 (1-2s)^3 h^3 + O(h^5)} \\ &\quad e^{(\mathbf{A_1}+\mathbf{A_2})sh + R_3 s^3 h^3 + O(h^5)} \\ &= e^{(\mathbf{A_1}+\mathbf{A_2})h + [2s^3 + (1-2s)^3] R_3 h^3 + O(h^5)} \end{aligned} \tag{4.29}$$

We can remove the term of h^3 by assigning the parameter s to satisfy $2s^3 + (1-2s)^3 = 0$, i.e. $s = 1/(2 - \sqrt[3]{2})$. Furthermore, if $\mathbf{A}$ is partitioned into n matrices where $\mathbf{A} = \mathbf{A_1} + \mathbf{A_2} + \cdots + \mathbf{A_n}$, the similar argument also works by

$$S_2(h) \equiv e^{\frac{\mathbf{A_1}h}{2}} \cdots e^{\frac{\mathbf{A_{n-1}}h}{2}} e^{\mathbf{A_n}h} e^{\frac{\mathbf{A_{n-1}}h}{2}} \cdots e^{\frac{\mathbf{A_1}h}{2}} \tag{4.30}$$

and $S(h) = S_2(sh)S_2((1-2s)h)S_2(sh)$ is still a fourth-order approximant for $e^{\mathbf{A}h}$.

With the previous approximation, we can derive the matrix exponential accurately by a product of exponent of matrices. It seems to require much time and work to compute a group of matrix exponentials. In actually, we can eliminate the full matrix multiplication operation in the numerical solution by this

approximation. The $e^{\mathbf{A}h}\mathbf{x}(T)$ in equation (4.24) can be rewritten as

$$e^{\mathbf{A}h}\mathbf{x}(T) = e^{\frac{\mathbf{A_1}sh}{2}} \dots e^{\mathbf{A_n}(1-2s)h} \dots e^{\frac{\mathbf{A_1}sh}{2}}\mathbf{x}(T).$$

If $e^{\mathbf{A_k}}$ is still a sparse matrix after an appropriate partition, then we can use sparse matrix-vector multiplication to compute $e^{\mathbf{A}h}\mathbf{x}(T)$ iteratively. We would like to mention that time complexity of sparse matrix-vector multiplication is low, and our experimental results show the number of partitioned matrices is limited. Therefore, by doing so, our method can derive matrix exponential efficiently.

Now, we present our strategy to partition $\mathbf{A}$ so that the evaluation of each $e^{\mathbf{A_k}h}$ is as efficient as possible and still keeps the sparsity. By the definition of matrix exponent in (4.25), the computation of powers of $\mathbf{A_k}$ becomes the essential part of the time complexity to evaluate $e^{\mathbf{A_k}h}$. Suppose $\mathbf{A_k} = [a_{ij}]_{1\leq i,j\leq m}$, we define $\mathbf{A_k}(\{i,j\})$ to be a 2×2 sub-matrix of $\mathbf{A_k}$ as

$$\mathbf{A_k}(\{i,j\}) \equiv \begin{pmatrix} a_{ii} & a_{ij} \\ a_{ji} & a_{jj} \end{pmatrix} \tag{4.31}$$

In our partition, we will choose $\mathbf{A_k}$ so that it can be decomposed into $\mathbf{A_k} = \mathbf{A_k}(\{i_1,j_1\}) + \dots + \mathbf{A_k}(\{i_s,j_s\})$ where the sets $\{i_1,j_1\},\dots,\{i_s,j_s\}$ are *pairwise disjoint*.

We illustrate our partition strategy by a simple 4-by-4 matrix:

$$\mathbf{A} = \begin{bmatrix} a_{11} & & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & & a_{44} \end{bmatrix},$$

which can be partitioned as: $\mathbf{A} = \mathbf{A_1} + \mathbf{A_2} + \mathbf{A_3}$, where

$$\mathbf{A_1} = \begin{bmatrix} a_{11} & & a_{13} & \\ & a_{22} & & a_{24} \\ a_{31} & & a_{33} & \\ & a_{42} & & a_{44} \end{bmatrix},$$

$$\mathbf{A_2} = \begin{bmatrix} & & & \\ & & a_{23} & \\ & a_{32} & & \\ & & & \end{bmatrix},$$

$$\mathbf{A_3} = \begin{bmatrix} & & & a_{14} \\ & & & \\ & & & \\ a_{41} & & & \end{bmatrix}.$$

The $\mathbf{A_1}$, $\mathbf{A_2}$ and $\mathbf{A_3}$ can be decomposed as

$$\begin{aligned} \mathbf{A_1} &= \mathbf{A_1}(\{1,3\}) + \mathbf{A_1}(\{2,4\}) \\ \mathbf{A_2} &= \mathbf{A_2}(\{2,3\}) \\ \mathbf{A_3} &= \mathbf{A_3}(\{1,4\}). \end{aligned}$$

In the above example, we can find that the sparsity of each $\mathbf{A_k}$ is still kept even after the computation of powers of matrix because every set is independent. Therefore, with this partition strategy, the $e^{\mathbf{A_k}h}$ is still a sparse matrix.

In addition, our partition strategy can provide a fast matrix exponential computation. The evaluation of exponent of $\mathbf{A_k}$ can be reduced into evaluation of exponent of its pairwise disjoint sets separately. Let $\mathbf{B}$ be any 2×2 matrix with Schur decomposition

$$\mathbf{B} = \mathbf{U}^* \Lambda \mathbf{U} = \mathbf{U}^* \begin{pmatrix} \lambda & \alpha \\ 0 & \mu \end{pmatrix} \mathbf{U}, \tag{4.32}$$

where $\mathbf{U}$ is unitary and λ, μ are eigenvalues of $\mathbf{B}$. The exponent of $\mathbf{B}$ can be

computed as

$$e^{\mathbf{B}} = \mathbf{U}^* e^{\Lambda} \mathbf{U} = \mathbf{U}^* \begin{pmatrix} e^{\lambda} & \frac{\alpha(e^{\lambda}-e^{\mu})}{\lambda-\mu} \\ 0 & e^{\mu} \end{pmatrix} \mathbf{U} \tag{4.33}$$

We develop Algorithm $MatrixPartition$ to partition $\mathbf{A}$ into $\mathbf{A_k}$'s with properties described above by using the union-find data structure. Line 1 in the pseudocode constructs a graph $G(V, E)$ with $V = \{1, 2, \ldots, m\}$ and $(u, v) \in E$ if and only if $a_{uv} \neq 0$. The $SizeBound$ in line 2 gives the independent block size in each $\mathbf{A_k}$ and usually we set it to two since the exponent of 2×2 matrices has the close form (4.33). During each iteration between lines 4 and 14, we create an $\mathbf{A_k}$ by adding elements to it greedily. The forest F initialized in line 6 represents our union-find data structure. The function $SetFriend(F, u, v, SizeBound)$ called in line 8 set u and v to be in the same equivalent class and returns $true$ if the size of new class is less than or equal to $SizeBound$, otherwise the function returns false. The matrix $\mathbf{E}_{uv}$ used in line 9 is an $m \times m$ matrix with one in position (u, v) and zero elsewhere. The correctness of our algorithm is clear and it runs in time $O(|E|)$ since the function $SetFriend$ takes constant time by a general implementation of union-find data structure.

4.5.3 Overall Numerical Integration Method

Now, we present an approximation for the numerical solution to avoid the matrix inverse of $\mathbf{A}$. The integrand of equation (4.23) can be approximated as piecewise linear [CL75]. Thus, the new approximated numerical solution can be expressed as

$$\mathbf{x}(T+h) = e^{\mathbf{A}h}\mathbf{x}(T) + e^{\mathbf{A}h}\frac{h}{2}\mathbf{u}(T) + \frac{h}{2}\mathbf{u}(T+h).$$

We combine the matrix exponential method with this approximation. The $e^{\mathbf{A}h}$ can be decomposed into a serial of sparse matrix exponential. Equation (4.34) shows our numerical integration method.

$$\mathbf{x}(T+h) = S(h)[\mathbf{x}(T) + \frac{h}{2}\mathbf{u}(T)] + \frac{h}{2}\mathbf{u}(T+h). \tag{4.34}$$

Algorithm 2: MatrixPartition

Input: An $m \times m$ sparse matrix $\mathbf{A} = [a_{ij}]$

Output: Matrices $\mathbf{A_1}, \ldots, \mathbf{A_n}$ with $\mathbf{A} = \mathbf{A_1} + \cdots + \mathbf{A_n}$

```
1  Construct graph G(V, E) from nonzero elements of A;
2  SizeBound=2;
3  k=1;
4  while E is not empty do
5      A_k = 0;
6      Set forest F with m nodes to be empty;
7      for each (u, v) ∈ E do
8          if SetFriend(F, u, v, SizeBound) then
9              A_k = A_k + a_uv E_uv;
10             Delete (u, v) from E;
11         end
12     end
13     k = k + 1;
14 end
```

We show the pseudo-code of our numerical integration method in Algorithm *NumericalIntegration*. In the algorithm, T_{total} is the time we want to simulate and *ArrayMatrixExp* is an array and store a group of matrix exponentials partitioned by Algorithm *MatrixPartition*. The for-loop performs vector and matrix multiplication, which is a fast computation.

Algorithm 3: NumericalIntegration

Input: A, $\mathbf{u}(t)$, $\mathbf{x}(0)$, h, and T_{total}

Output: $\mathbf{x}(t)$

```
1  Partition A by Algorithm MatrixPartition;
2  ArrayMatrixExp = S(h);
3  t=0;
4  while t < T_total − h do
5      v = x(t) + (h/2)u(t);
6      for each matrix M in ArrayMatrixExp do
7          v = Mv;
8      end
9      x(t + h) = v + (h/2)u(t + h);
10     t = t + h;
11 end
```

Suppose the maximum number of non-zero elements of the matrices in *ArrayMatrixExp* is m, and the number of input sources is n. The complexity of our algorithm of each time step is $O(|ArrayMatrixExp|mn)$. Though our method needs to do $|ArrayMatrixExp|$ times more matrix multiplication than forward Euler, we can use larger step size without the stability issue and our method has better performance in overall.

4.5.4 Stability and Error Analysis

In equation (4.34), $S(h)$ must be less than 1 to guarantee the stability of our method. Because the $S(h)$ is the matrix exponential, the constraint can always

be satisfied when the eigenvalue of $\mathbf{A}$ is negative that usually is true in electronic circuits. Thus, our method has no stability issue, and is A-stable.

For the error analysis, we estimate the numerical error of our method to the numerical solution. The error is the difference between equation(4.34) and (4.24). We assume $S(h) = e^{\mathbf{A}h} + O(h^5)$, and $\mathbf{u}(T) \approx \mathbf{u}(T+h)$. Then, by equation (4.25), we expand the $e^{\mathbf{A}h}$ of $\mathbf{u}$ term in both equations and rewritten them as:

$$(e^{\mathbf{A}h} - \mathbf{I})\mathbf{A}^{-1}\mathbf{u}(T) = (h + \frac{\mathbf{A}h^2}{2!} + \frac{\mathbf{A}^2h^3}{3!} + \dots)\mathbf{u}(T)$$
$$S(h)\frac{h}{2}\mathbf{u}(T) + \frac{h}{2}\mathbf{u}(T+h) = (h + \frac{\mathbf{A}h^2}{2} + \frac{\mathbf{A}^2h^3}{2 \cdot 2!} + \dots)\mathbf{u}(T)$$

The difference of $\mathbf{x}(T)$ and $\mathbf{u}$ terms are $O(h^5)$ and $O(h^3)$. Therefore, the overall numerical error of our method is $O(h^3)$.

4.5.5 Comparison with Euler Method

We implemented our numerical integration method by matrix exponent and then compared it with forward and backward Euler method. All methods are written in MATLAB except the part of solving matrix in backward Euler method, for which we use the KLU package [Dav06]. The experiments are performed on a Linux machine with an Intel Xeon 3.0GHz processor and 16GB memory.

We use different on-chip power network designs as our benchmark circuits. They are of mesh structure and the R, L and C parameters of the power network are 0.87Ω/mm, $0.41n$H/mm, and $1.08p$F/mm, respectively. We also include the package model in the designs where the lumped R and L are $10m\Omega$ and $0.2p$H.

First, we compare the stability of our method and forward Euler method. The simulation results of a power network with $160,000$ nodes are shown in Figure 4.10. The step size $h = 5 \times 10^{-13}$ is used for both methods. However, this step size is too large for forward Euler and cause divergence. Then, we applied $h = 1 \times 10^{-14}$ for forward Euler. The runtime of our method is 273.85 seconds. In contrast, forward Euler method takes 722.11 seconds because it needs smaller time step size to maintain the stability. Although the computation of forward Euler at each time step is faster than our method which needs more matrix multiplications, the stability issue damages this advantage of forward Euler.

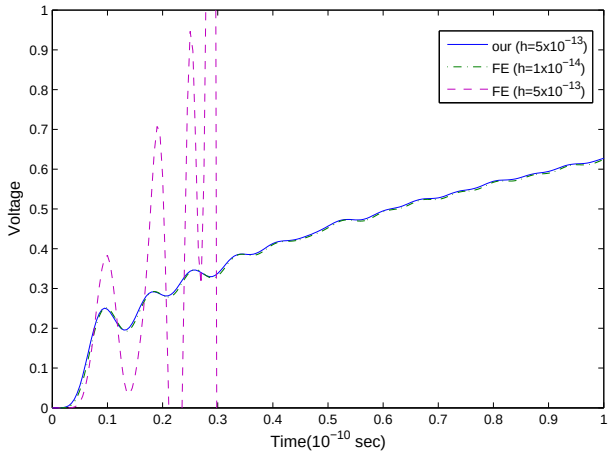


Figure 4.10: Comparison between our method and forward Euler

Figure 4.11 shows the result with different step sizes of the same power network design by our method. The step sizes varies from 1×10^{-12} to 1×10^{-14}. We can find that our method is stable even under a wide range of step sizes. The error of our method comes from the matrix exponential computation and the approximation error of equation (4.34). Figure 4.12 shows a close look of the results. The results of $h = 1 \times 10^{-13}$ and $h = 1 \times 10^{-14}$ are almost the same. However, the error between $h = 1 \times 10^{-12}$ and $h = 1 \times 10^{-13}$ are noticeable. This is because our error is proportional to the powers of the time step size.

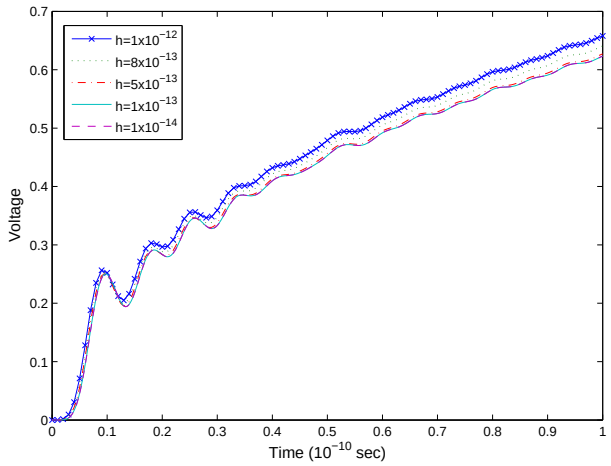


Figure 4.11: Our method with different step sizes

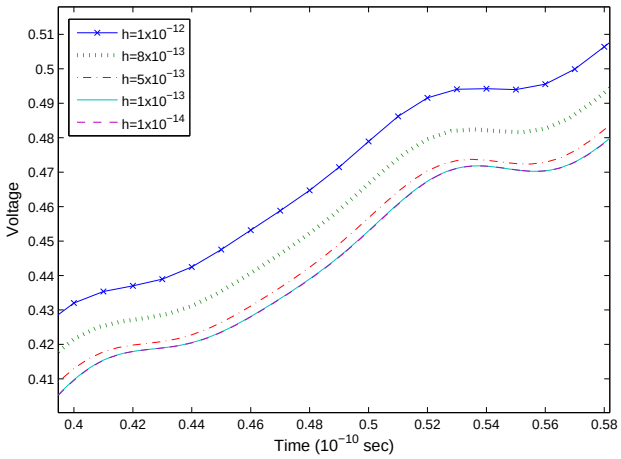


Figure 4.12: A zoom-in view of figure 4.11

Finally, we compare our method with backward Euler method. Both methods use the same time step size of 5×10^{-14}. Table 4.1 shows the runtime for different power network designs. The size of the design is shown in the second column. Column 3 and 4 show the runtime of our method and backward Euler. The last column shows the speedup over backward Euler. The number of partitioned matrix is 7 for all cases. This is because the matrix of the circuit is sparse. We only need a few matrices to represent the matrix **A**. The results show our method is two times faster than backward Euler on average. Also, our method is more scalable. For the designs with 3.2M and 4M nodes, KLU is failed on factorization. Figure 4.13 shows the detail of results of our method and backward Euler of power network design 1. As we can see, our method is more accurate than backward Euler under the same step size. The result of backward Euler is close to ours only when the time step goes down to 1×10^{-14}.

4.6 Nonlinear Programming Solver

We adopt the *sequential quadratic programming (SQP)* algorithm to solve our non-linear programming problem 4.6 and 4.7. SQP algorithm is an iterative method which utilizes Newton's method and Karush-Kuhn-Tucker (KKT) conditions to derive a general framework to handle non-linear optimization problems.

Table 4.1: Runtime of Our Method and Backward Euler Method

Circuit	# of Nodes	Our(sec)	BE(sec)	Speedup
Power network 1	160K	1602.68	3617.31	2.25
Power network 2	250K	2350.27	4947.41	2.11
Power network 3	360K	3235.11	6707.04	2.07
Power network 4	640K	5505.72	11654.24	2.12
Power network 5	1M	8364.33	15724.31	1.87
Power network 6	1.4M	11869.03	29072.41	2.4
Power network 7	3.2M	27083.53	fail	N/A
Power network 8	4M	33258.37	fail	N/A

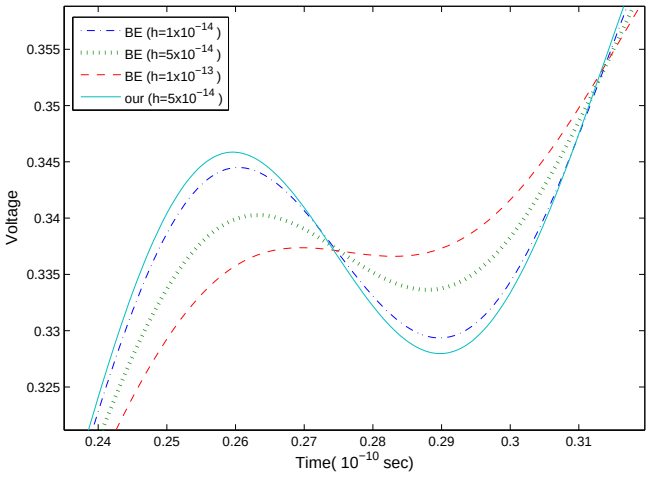


Figure 4.13: Results of Power Network Design 1

We first consider the following formulation of a optimization problem with only equality constraints:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & f(x) \\ \text{s.t.} \quad & c(x) = 0 \end{aligned} \tag{4.35}$$

For an unconstraint optimization problem, we can use the first-order optimality conditions $\nabla f(x) = 0$ to solve a set of first-order optimal points x^*. In the case of optimization problem with equality constraints as 4.35, KKT conditions state that if x^* is a local minimum of the problem 4.35 with regular conditions, then it satisfies:

$$\nabla \mathscr{L}(x, \lambda) = 0,$$

where $\mathscr{L}(x, \lambda)$ is the Lagrange duality function of problem 4.35 with definition:

$$\mathscr{L}(x, \lambda) = f(x) - \lambda^T c(x)$$

We define $g(x) \equiv \nabla f(x)$, $J(x)$ as the matrix of the Jacobian of constraints and $F(x, \lambda) \equiv \nabla \mathscr{L}(x, \lambda)$. The KKT condition can be written in the following matrix form:

$$F(x, \lambda) = \begin{pmatrix} g(x) - J(x)^T \lambda \\ -c(x) \end{pmatrix} = 0 \tag{4.36}$$

We utilize the Newton's method to solve the equation 4.36. In the k^{th} iteration of the Newton's method, the following equation will be computed to obtain x_{k+1}, λ_{k+1} from x_k, λ_k.

$$\begin{aligned} \begin{pmatrix} x_{k+1} \\ \lambda_{k+1} \end{pmatrix} &= \begin{pmatrix} x_k \\ \lambda_k \end{pmatrix} + \begin{pmatrix} \Delta x_k \\ \Delta \lambda_k \end{pmatrix}, \text{ where} \\ \begin{pmatrix} \Delta x_k \\ \Delta \lambda_k \end{pmatrix} &= \frac{-F(x, \lambda)}{F'(x, \lambda)} \end{aligned} \tag{4.37}$$

Let $H(x, \lambda)$ be the Hessian matrix of $\mathscr{L}(x, \lambda)$ on x, i.e. $H(x, \lambda) = \nabla_x x^2 \mathscr{L}(x, \lambda)$. We can represent the differentiation of F as the following formula:

$$F'(x, \lambda) = \begin{pmatrix} H(x, \lambda) & -J(x)^T \\ -J(x) & 0 \end{pmatrix} \tag{4.38}$$

Let H_k, J_k, g_k and c_k be the evaluations of functions $H(x,\lambda), J(x), g(x)$ and $c(x)$ on x_k and λ_k. The iteration steps Δx_k and $\Delta\lambda_k$ in 4.37 can be computed by solving the following equations:

$$\begin{pmatrix} H_k & -J_k^T \\ -J_k & 0 \end{pmatrix} \begin{pmatrix} \Delta x_k \\ \Delta\lambda_k \end{pmatrix} = -\begin{pmatrix} g_k - J_k^T\lambda \\ -c_k \end{pmatrix} \tag{4.39}$$

We scale the equation 4.39 and obtain a equivalent symmetric form:

$$\begin{pmatrix} H_k & J_k^T \\ J_k & 0 \end{pmatrix} \begin{pmatrix} \Delta x_k \\ -\Delta\lambda_k \end{pmatrix} = -\begin{pmatrix} g_k - J_k^T\lambda_k \\ c_k \end{pmatrix} \tag{4.40}$$

The next objective is to represent the equation 4.39 as a first-order optimality conditions of an optimization problem by using the KKT conditions again. We notice that $\lambda_{k+1} = \lambda_k + \Delta\lambda_k$ and the equation 4.40 can be simplified as:

$$\begin{pmatrix} H_k & J_k^T \\ J_k & 0 \end{pmatrix} \begin{pmatrix} \Delta x_k \\ -\lambda_{k+1} \end{pmatrix} = -\begin{pmatrix} g_k \\ c_k \end{pmatrix} \tag{4.41}$$

We consider the following quadratic programming problem on d whose first-order optimality conditions are exactly the equation 4.41, and $\Delta x_k, \lambda_{k+1}$ are the primal and dual solution of the quadratic program.

$$\begin{aligned} \min_{d\in\mathbb{R}^n} \quad & f(x_k) + g_k^T d + \frac{1}{2} d^T H_k d \\ \text{s.t.} \quad & c_k + J_k^T d = 0 \end{aligned} \tag{4.42}$$

The similar deduction can be applied for the general optimization problem with inequality constraints. We write a formulation of these problems as follows:

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & f(x) \\ \text{s.t.} \quad & c(x) = 0 \\ & h(x) >= 0 \end{aligned} \tag{4.43}$$

The Lagrange duality function of problem 4.43 is:

$$\mathscr{L}(x,\lambda,\mu) = f(x) - \lambda^T c(x) - \mu^T h(x)$$

At the k^{th} iteration with current primal solution x_k and dual solutions λ_k, μ_k, the SQP algorithm will update them based on the primal solution Δx_k and dual solutions λ_{k+1}, μ_{k+1} of the following quadratic optimization problem:

$$\begin{aligned} \min_{d\in\mathbb{R}^n} \quad & \mathscr{L}(x_k,\lambda_k,\mu_k) + \nabla\mathscr{L}(x_k,\lambda_k,\mu_k)^T d + \frac{1}{2} d^T \nabla^2_{xx}\mathscr{L}(x_k,\lambda_k,\mu_k) d \\ \text{s.t.} \quad & c(x_k) + \nabla c(x_k)^T d = 0 \\ & h(x_k) + \nabla h(x_k)^T d \geq 0 \end{aligned} \tag{4.44}$$

4.7 Experimental Results

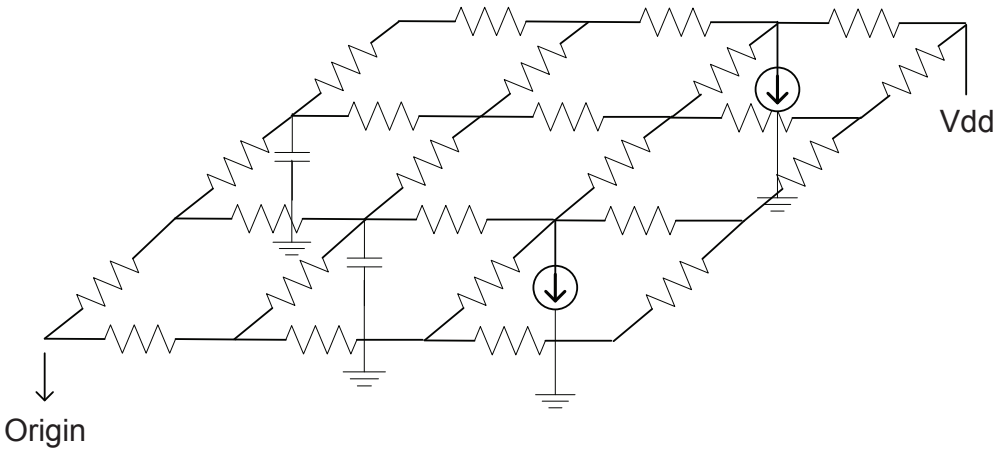


Figure 4.14: Power Network Model

Figure 4.14 shows the power network model for our experiments including resistances, capacitances and current sources. The V_{dd} are placed at the top-right corner of the power network with voltage $1V$. The controlled-ESRs connect adjacent network nodes and Decaps connect network nodes with the ground. Time-varying current sources are placed at some nodes, characterizing the supply current for active circuit instances. The waveform of current sources is assumed as a piecewise linear function.

We implement our algorithms for simulation and SQP optimization by Matlab on a machine with 3.0GHz Intel Xeon processor and 4GB memory. We consider

a power network with die size $0.4mm$ by $0.4mm$ and grid size 200 by 200. The time range T and unit time ∇t are set to be $1ns$ and $5ps$ respectively. Four current sources $I_1(t), I_2(t), I_3, I_4(t)$ are placed on the network at the positions $(90, 0)$,$(90, 90)$,$(190, 0)$ and $(190, 90)$ respectively with the transition time $t_r = 20ps$ to increase/decrease 1A. The current sources satisfy the following duty cycle constraints and Group Magnitude Constraints:

- Time range $[0ps, 200ps)$:

$$0 \leq I_k(t) \leq 1.0, \;\; 1 \leq k \leq 5$$
$$I_1(t) + I_2(t) \leq 1.5,$$
$$I_1(t) + I_2(t) + I_3(t) + I_4(t) + I_5(t) \leq 3.0.$$

- Time range $[200ps, 400ps)$:

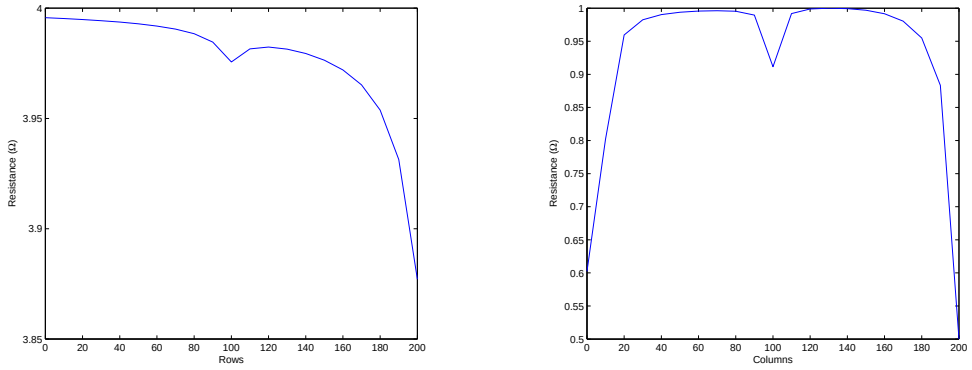
$$0 \leq I_k(t) \leq 2.0, \;\; 1 \leq k \leq 5$$
$$I_3(t) + I_4(t) \leq 3.0,$$
$$I_1(t) + I_2(t) + I_3(t) + I_4(t) + I_5(t) \leq 6.0.$$

- Time range $[400ps, 600ps)$:

$$0 \leq I_k(t) \leq 1.0, \;\; 1 \leq k \leq 5$$
$$I_1(t) + I_3(t) \leq 1.5,$$
$$I_1(t) + I_2(t) + I_3(t) + I_4(t) + I_5(t) \leq 3.0.$$

- Time range $[600ps, 800ps)$:

$$0 \leq I_k(t) \leq 3.0, \;\; 1 \leq k \leq 5$$
$$I_2(t) + I_4(t) \leq 4.0,$$
$$I_1(t) + I_2(t) + I_3(t) + I_4(t) + I_5(t) \leq 9.0.$$



(a) Optimal row resistances.

(b) Optimal column resistances.

Figure 4.15: Optimal resistances distribution.

- Time range $[800ps, 1000ps]$:

$$0 \leq I_k(t) \leq 2.0, \;\; 1 \leq k \leq 5$$
$$I_1(t) + I_2(t) \leq 3.5,$$
$$I_3(t) + I_4(t) \leq 3.5,$$
$$I_1(t) + I_2(t) + I_3(t) + I_4(t) + I_5(t) \leq 6.0.$$

We optimize the voltage violation area with $V_{min} = -12V$ at the origin $(0, 0)$ by tuning resistances and capacitances. We assume resistors at the same row or column have the same value with respect to the general technical constraint. The resistors with the horizontal direction are initialized as 4.0Ω and tuned in the range $[2.0\Omega, 6.0\Omega]$. The resistors with the vertical direction are initialized as 1.0Ω and tuned in the range $[0.5\Omega, 2.0\Omega]$. The capacitors connected with nodes are initialized as $1pF$ and tuned in the range $[0.1pF, 10pF]$.

Figure 4.15 shows the optimal resistance distribution for rows and columns respectively. We observe that resistances become smaller when it is near the current and voltage sources. Since the current sources locate at the top-left corner, the row resistors are roughly decreasing except for the row 90 corresponding to the current sources at $(90, 0)$ and $(90, 90)$. The three local minima of column resistors reflect the current sources $(90, 0)/(190, 0)$, $(90, 90)/(190, 90)$ and the Vdd

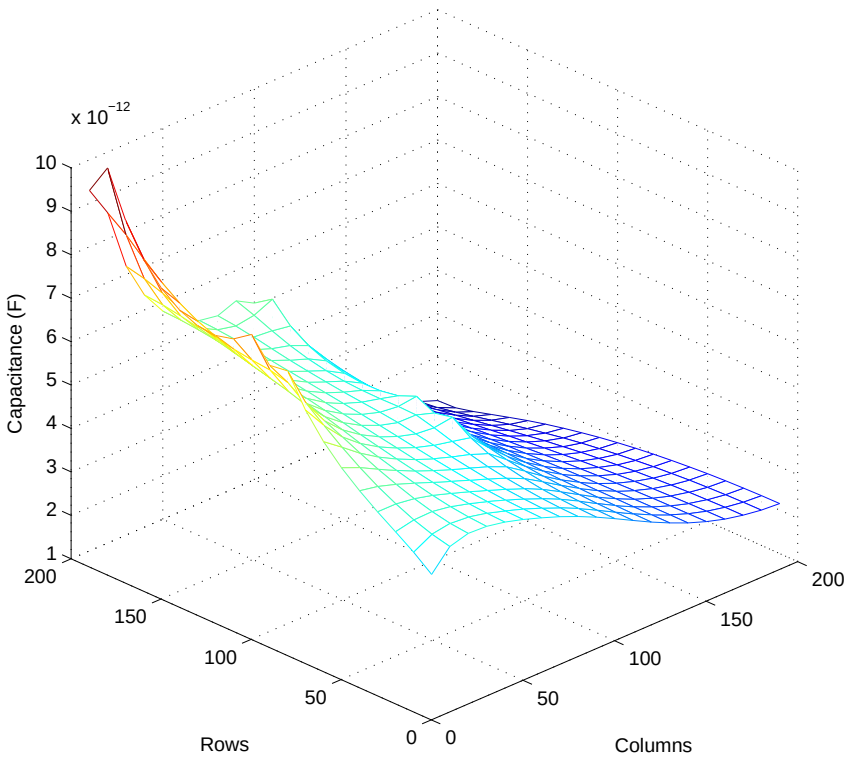
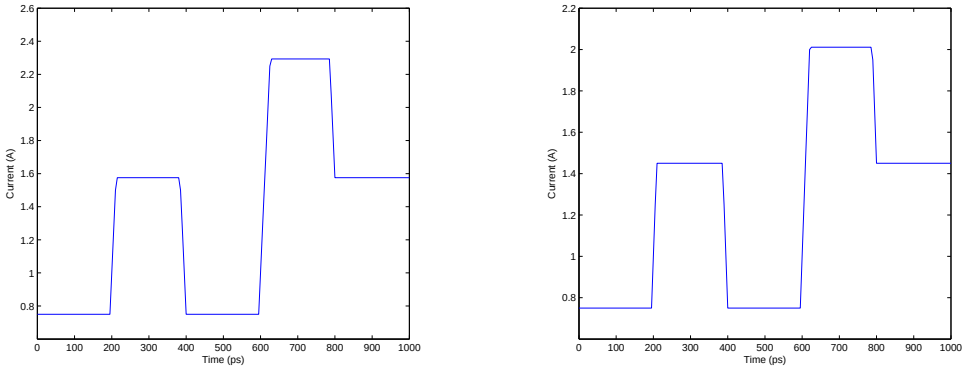
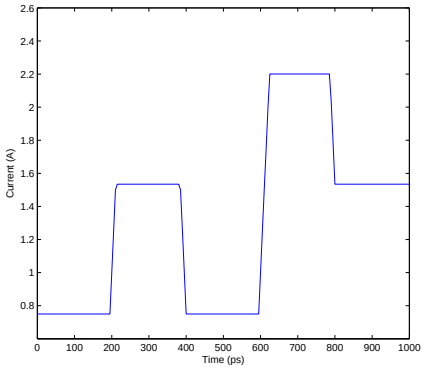


Figure 4.16: Optimal capacitance distribution

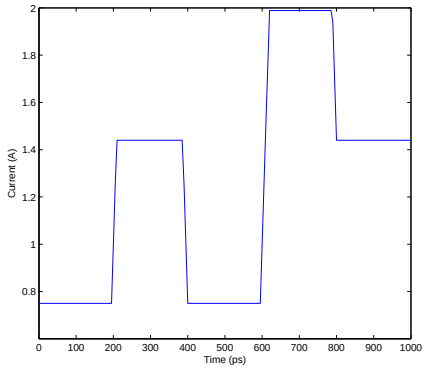


(a) Current source at $(90, 0)$.

(b) Current source at $(90, 90)$.



(c) Current source at $(190, 0)$.



(d) Current source at $(190, 90)$.

Figure 4.17: The worst-case profiles of current sources.

correspondingly. Figure 4.16 illustrates the optimal capacitance at each node. Generally, the capacitances with location near the current sources are larger. The smallest capacitor locates at the Vdd position.

Figure 4.17 depicts the worst case current profile for each current source. The trend of different current sources is nearly same because of the constraints on five duty cycles. The currents for each duty cycle are distributed into four current sources in order to generate the worst case violation area of voltage. Figure 4.18 compares the voltage waveform at the origin before and after optimization. We observe that the violation area has been reduced significantly. Through the optimal

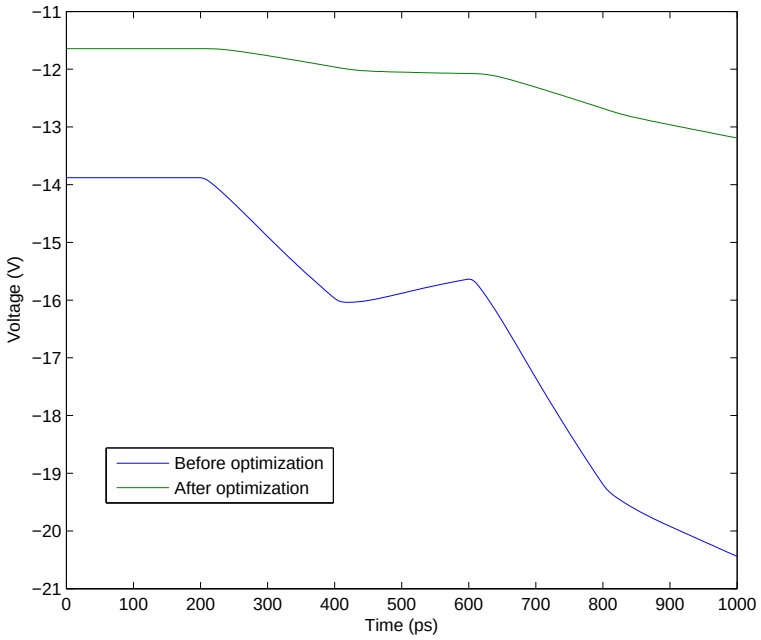


Figure 4.18: Voltage profile at the origin.

distribution of resistors and capacitors, we also improve the worst voltage drop by 33.80% relative to the uniform distribution.

Chapter 4, in part, is currently being prepared for submission for publication of the material. Peng Du; Shih-Hung Weng; Xiang Hu; Chung-Kuan Cheng. The dissertation author was the primary investigator and author of this material. The section four, in part, is a reprint of the material as it appears in Asia and South Pacific Design Automation Conference(ASPDAC) 2010. Xiang Hu; Wenbo Zhao; Peng Du; Amirali Arani; Chung-Kuan Cheng, IEEE, 2010. The dissertation author was a cooperative author of this paper. The section five, in part, is a reprint of the material as it appears in IEEE Symposium on Circuits and Systems(ISCAS) 2011. Shih-Hung Weng; Peng Du; Cheng-Kuan Cheng, IEEE, 2011. The dissertation author was a cooperative author of this paper.

Chapter 5

Conclusion

In this thesis, we develop a whole flow of power network verification and optimization at planning stages. We take the worst case voltage drop (or voltage violation area) as the objective since it degrades the timing performance and reliability of circuits. The verification problem asks for predicting the voltage noise subject to certain current constraints. The optimization problem asks for tuning circuit parameters (e.g. resistances and capacitances) in order to minimize the voltage noise.

Based on the prior knowledge of impulse response, we propose a dynamic programming algorithm to predict the worst case voltage drop and extend it to multiple current sources with hierarchical constraints. More algorithms including network flow and submodular flow are adopted to handle the cases with more complicated current constraints than hierarchical constraints. These methods are much more efficient than the trivial linear programming formulation of the verification problem.

If the information about impulse response is not available, we devise a framework for solving both the verification problem and optimization problem alternatively by computing the sensitivity via adjoint network approach. Fast circuit simulation methods are developed in order to run the non-linear program solver efficiently. This frame work can handle complicated current constraints including duty-cycle constraints, group magnitude constraints and transition time constraints.

For the pure resistance network, we reduce the optimization problem into a convex programming problem by relating the worst case voltage drop with the effective resistance in the network. We adopt a fast Krylov space method to evaluate the effective resistance and derive a closed-form formula to update the derivative of effective resistance relative to resistances gradually. The solver of convex programming is much faster than that of non-linear programming and we can obtain the global optimal solution for this problem.

5.1 Power Network Verification

We divide the power network verification problem into two cases where single current source and multiple current sources are assumed respectively. For both cases, we handle load currents with non-zero transition time and require only a one-time simulation of the PDS impulse response. In Chapter 2.1, we propose a dynamic programming algorithm to generate worst case noise and accelerate it using a Knuth-Yao Quadrangle Inequalities Speedup. With the proposed approach, the worst-case noise behavior with respect to the transition time has been studied. The worst-case noise decreases with the transition time, which demonstrates that assuming a zero transition time will lead to an overly pessimistic worst-case noise prediction. In Chapter 2.2, we extend the dynamic programming algorithm into multiple current sources satisfying the hierarchical constraints. Furthermore, we argue that the hierarchical constraints can be treated as a special case of submodular polyhedron constraint where the dynamic programming algorithm still works. In order to handle the case where current sources have more general magnitude and slope constraints, we introduce to methods via network flow and submodular flow techniques which are much faster than naive linear programming solution.

5.2 Power Network Optimization

For the resistance power network, we propose a power grid sizing method to minimize the worst voltage drop over all test locations and current source dis-

tributions. We reduce the original problem into a convex programming problem whose objective is to minimize the maximum effective resistance between the current entry node and all current exit nodes under the constraint of constant total wire area. In order to solve the convex programming problem efficiently, we adopt a Krylov space method to evaluate the effective resistances simultaneously and deduce a simple formula to update the derivative of effective resistance relative to perturbation of wire widths gradually. Experimental results show that our method can achieve up to 40% improvement for regular 2D grids and 7.32% improvement for a practical power grid with only top two layers tunable over uniform sizing. In addition, the proposed optimization method can also be applied to power grids in the real world, which are required to have small effective resistances among power stations for reducing the power losses during long distance transmission.

For the dynamic power distribution network, we develop an adjoint network approach to perform early stage verification and optimization. The method assume multiple current sources with duty-cycle constraints, group magnitude constraints and transition time constraints which makes the prediction more realistic. Moreover, the worst violation area is optimized by allocating decoupling capacitors and controlled equivalent series resistors. The flow handles the verification and optimization problems alternatively where the adjoint network technique is used to evaluate the sensitivity of different objectives. Finally, the sequential quadratic programming method is adopted for solving the proposed non-linear programming problem. We also propose two fast circuit simulation methods including adaptive DFT and matrix exponential in order to obtain the properties of original and adjoint network efficiently.

Bibliography

[AMO93] R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[BV04a] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[BV04b] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[Car10] P. Carrier. Diagonal of inverse of real symmetric and complex diagonal matrices using Lanczos. In *Personal notes*, February 2010.

[CL75] L. O. Chua and P. M. Lin. *Computer Aided Analysis of Electric Circuits: Algorithms and Computational Techniques*. Prentice-Hall, 1975.

[CLRS01] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms (Second ed.)*. MIT Press and McGraw-Hill, 2001.

[Dav06] T. A. Davis. *Direct Method for Sparse Linear Systems*. SIAM, 2006.

[DHW+10] P. Du, X. Hu, S. H. Weng, A. Shayan, X. Chen, A. E. Engin, and C. K. Cheng. Worst-Case Noise Prediction With Non-zero Current Transition Times for Early Power Distribution System Verification. In *IEEE International Symposium on Quality Electronic Design*, 2010.

[DR93] A. Devgan and R. A. Rohrer. Event driven adaptively controlled explicit simulation of integrated circuits. In *ICCAD '93: Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design*, pages 136–140, 1993.

[EGG88] D. Eppstein, Z. Galil, and R. Giancarlo. Speeding up Dynamic Programming. In *In Proc. 29th Symp. Foundations of Computer Science*, pages 488–496, 1988.

[FCN08] I. A. Ferzli, E. Chiprout, and F. N. Najm. Verification and Co-design of the Package and Die Power Delivery System Using Wavelets. In *Electrical Performance of Electronic Packaging*, pages 7–10, 2008.

[FLH+04] J. Fu, Z. Luo, X. Hong, Y. Cai, S. Tan, and Z. Pan. A fast decoupling capacitor budgeting algorithm for robust on-chip power delivery. In *Asia and South Pacific Design Automation Conference*, pages 505–510, January 2004.

[FNK07] I. A. Ferzli, F. N. Najm, and L. Kruze. A Geometric Approach for Early Power Grid Verification Using Current Constraints. In *ACM/IEEE International Conference on Computer-Aided Design*, pages 40–47, November 2007.

[Fuj05] S. Fujishige. *Submodular Functions and Optimization*. Elsevier Science, 2005.

[GB11] M. Grant and S. Boyd. Cvx: Matlab software for disciplined convex programming, Feb 2011. `http://cvxr.com/cvx/`.

[GBS06] A. Ghosh, S. Boyd, and A. Saberi. Minimizing effective resistance of a graph. In *Proceedings of the 17th International Symposium on Mathematical Theory of Networks and Systems*, 2006.

[GK06] P. Gupta and A.B. Kahng. Efficient design and analysis of robust power distribution meshes. In *VLSI Design, 2006. Held jointly with the 5th International Conference on Embedded Systems and Design*, page 6. IEEE, 2006.

[GN09] N. H. Abdul Ghani and F. N. Najm. Fast Vectorless Power Grid Verification Using an Approximate Inverse Technique. In *ACM/IEEE Design Automation Conference*, pages 184–189, July 2009.

[HS05] N. Hatano and M. Suzuki. Finding exponential product forumulas of higher orders. *Lecture Notes in Physics*, 679:36–68, 2005.

[HZZ+09] X. Hu, W. Zhao, Y. Zhang, A. Shayan, C. Pan, A. E. Engin, and C. K. Cheng. On the Bound of Time-Domain Power Supply Noise Based on Frequency-Domain Target Impedance. In *System Level Interconnect Prediction Workshop*, pages 69–76, July 2009.

[JC] J. K. Johnson and M. Chertkov. A majorization-minimization approach to design of power transmission networks.

[KN03] D. Kouroussis and F. N. Najm. A Static Pattern-Independent Technique for Power Grid Voltage Integrity Verification. In *ACM/IEEE Design Automation Conference*, pages 99–104, June 2003.

[LQT+06] H. Li, Z. Qi, S. Tan, L. Wu, Y. Cai, and X. Hong. Partitioning-based approach to fast on-chip decap budgeting and minimization. *IEEE Transactions on CAD*, 25(11):2402–2412, November 2006.

[PMF08] M. Popovich, A. V. Mezhiba, and E. G. Friedman. *Power Distribution Networks with On-Chip Decoupling Capacitors*. Springer, 2008.

[PSKF08] M. Popovich, M. Sotman, A. Kolodny, and E. G. Friedman. Effective Radii of On-Chip Decoupling Capacitors. *IEEE Transaction on Very Large Scale Integration Systems*, 16(7):894–907, July 2008.

[S.B06] S.Boyd. Convex Optimization of Graph Laplacian Eigenvalues. In *Proceedings International Congress of Mathematicians*, pages 1311–1319, 2006.

[SGS03] H. Su, K. Gala, and S. Sapatnekar. Analysis and optimization of structured power/ground networks. *IEEE Transactions on CAD*, 22(11):1533–1544, November 2003.

[SPKF05] M. Sotman, M. Popovich, A. Kolodny, and E. G. Friedman. Leveraging Symbiotic On-Die Decoupling Capacitance. In *IEEE Top. Meet. Elect. Performance of Electronic Packaging*, pages 111–114, October 2005.

[SSN03] H. Su, S. Sapatnekar, and S. Nassif. Optimal decoupling capacitor sizing and placement for standard-cell layout designs. *IEEE Transactions on CAD*, 22(4):428–436, April 2003.

[SVGY01] S.Boyd, L. Vandenberghe, A. El Gamal, and S. Yun. Design of Robust Global Power and Ground Networks. In *Proceedings ACM Symposium on Physical Design*, pages 60–65. ACM, 2001.

[TWKA06] Y. Tanji, T. Watanabe, H. Kubota, and H. Asai. Large scale rlc circuit analysis using RLCG-MNA formulation. In *DATE '06: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 45–46, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association.

[VBG98] L. Vandenberghe, S. Boyd, and A. El Gamal. Optimizing dominant time constant in RC circuits. *Computer Aided Design, IEEE Transactions on*, 17(2):110–125, 1998.

[WC02] T.Y. Wang and C.C.P. Chen. Optimization of the power/ground network wire-sizing and spacing based on sequential network simplex algorithm. In *Proceedings of the International Symposium on Quality Electronic Design*, pages 157–162. IEEE, 2002.

[WMS05] K. Wang and M. Marek-Sadowska. On-chip power-supply network optimization using multigrid-based technique. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(3):407–417, 2005.

[Yao80] F. Frances Yao. Efficient Dynamic Programming Using Quadrangle Inequalities. In *Twelfth Annual ACM Symposium on Theory of Computing*, pages 429–435, 1980.