AN ADAPTIVE LINEARIZATION METHOD FOR
A CONSTRAINT SATISFACTION PROBLEM
IN SEMICONDUCTOR DEVICE
DESIGN OPTIMIZATION

DISSERTATION

Presented to the Graduate Council of the
University of North Texas in Partial
Fulfillment of the Requirements

For the Degree of

DOCTOR OF PHILOSOPHY

By

Chih-Hui Chang, M.S.
Denton, Texas
May, 1999

The device optimization is a very important element in semiconductor technology advancement. Its objective is to find a design point for a semiconductor device so that the optimized design goal meets all specified constraints. As in other engineering fields, a nonlinear optimizer is often used for design optimization. One major drawback of using a nonlinear optimizer is that it can only partially explore the design space and return a local optimal solution. This dissertation provides an adaptive optimization design methodology to allow the designer to explore the design space and obtain a globally optimal solution. One key element of our method is to quickly compute the set of all feasible solutions, also called the acceptability region. We described a polytope-based representation for the acceptability region and an adaptive linearization technique for device performance model approximation. These efficiency enhancements have enabled significant speed-up in estimating acceptability regions and allow acceptability regions to be estimated for a larger class of device design tasks. Our linearization technique also provides an efficient mechanism to guarantee the global accuracy of the computed acceptability region. To visualize the acceptability region, we study the orthogonal projection of high-dimensional convex polytopes and propose an output sensitive algorithm for projecting polytopes into two dimensions.
ACKNOWLEDGEMENTS

I would like to thank my supervisor, Professor Weiping Shi, for his constant support and encouragement. Without his patient guidance, this thesis would have been impossible.

I am grateful to Professor Steve Tate, Professor Tom Jacob and Professor Sharad Saxena, for their careful reading and valuable feedback. Thanks are due to P.K. Mozumder, Sharad Saxena and Richard Burch who have made my summer student experience at Texas Instruments so enjoyable.

I am also very thankful to my parents for their generous and unconditional support. Finally, I would like to thank my wife Pin-Chieh and my daughter Stephanie for standing by me and making my life so joyful.
# TABLE OF CONTENTS

LIST OF TABLES                             vi

LIST OF ILLUSTRATIONS                    vii

1 INTRODUCTION                          1
   1.1 Semiconductor Device Fabrication Process        1
   1.2 Device Design Optimization                     5
       1.2.1 The Device Decomposition Problem          6
   1.3 Related Work                                  9
   1.4 Contributions                                18
   1.5 Outline of the Dissertation                   19

2 MODEL LINEARIZATION AND REPRESENTATION  21
   2.1 From Piecewise Constant to Piecewise Linear   22
   2.2 Linear Approximation for a Quadratic Function 27
       2.2.1 Linearizing a square term $x_i^2$         28
       2.2.2 Linearizing a product term $x_i x_j$      32
       2.2.3 Linearizing a quadratic function          39
   2.3 Error Measurement and Designable Space Partitioning 41
       2.3.1 An improvement to reduce the number of space subdivisions 45
   2.4 An Adaptive Linearization Algorithm           49
   2.5 Binary Space Partition Tree                   54
3 ACCEPTABILITY REGION COMPUTATION

3.1 A New Polytope Representation for Acceptability Subregion .......................... 57
3.2 Acceptability Region in Binary Space Partitioning Tree ................................ 62
3.3 Acceptability Region Visualization ................................................................. 65
3.4 Projection of a Convex Polytope ................................................................. 69
   3.4.1 A New Algorithm for The Two-Dimensional Case ................................... 72

4 DESIGN OPTIMIZATION WITH ACCEPTABILITY REGIONS ................................. 76

4.1 Global Optimization with Acceptability Regions ........................................... 77
4.2 Extracting Design Information from Acceptability Regions .............................. 78
4.3 Design with Acceptability Regions ................................................................. 80
4.4 Consistent Acceptability Regions After Editing ............................................ 81
4.5 Handling More Difficult and High Dimensional Problems .............................. 82

5 IMPLEMENTATION AND EXPERIMENTAL RESULTS ............................................. 84

5.1 General Function ............................................................................................... 84
5.2 Quadratic Function ........................................................................................... 86
5.3 Visualization ....................................................................................................... 91
5.4 Summary ............................................................................................................ 94

6 CONCLUSIONS .................................................................................................... 95

6.1 Major Results .................................................................................................... 95
6.2 Future Research Extensions ............................................................................. 97
6.3 Summary ........................................................................................................... 97

REFERENCES ......................................................................................................... 99
LIST OF TABLES

1.1 Some typical process steps .............................................. 3

5.1 An experiment result on a general function case ...................... 85

5.2 An experiment result on a quadratic function case ..................... 91
LIST OF ILLUSTRATIONS

1.1 IC process flow ........................................ 1
1.2 A p-n junction fabrication process sequence ............. 4
1.3 A NMOS transistor ....................................... 4
1.4 An example of device decomposition ......................... 9
1.5 An example of ternary relation construction from a clause .... 15
1.6 The ternary function transformation and the relaxed equality representation introduce inaccuracy. .................. 16

2.1 Comparison of piecewise constant and piecewise linear representation 23
2.2 Adaptive Piecewise Linearization .......................... 24
2.3 The data structure for the piecewise linear model ............. 24
2.4 Linearization of the square term, $x_i^2$ ...................... 29
2.5 Linearization of the product term, $x_i x_j$ .................... 35
2.6 Relation between the subdomain $R$ and two lines $x_i = b$ and $x_j = a$ . 38
2.7 Error in piecewise constant representation .................. 42
2.8 An example for BSP tree .................................. 55
2.9 An example BSP tree for piecewise linear representation .... 56

3.1 Three cases of feasible subregion for a type 3 subdivision .... 60
3.2 Shadow and slice of a three dimensional acceptability region .... 67
3.3 Using linear programming to find polygon vertex ............ 73
3.4 An example for the new two-dimensional projection algorithm .... 74

4.1 An adaptive design optimization paradigm ................... 77
4.2 Showing the performance approximation along with the acceptability region ............................................. 80

5.1 An acceptability region for specification 1 on $x_7-x_6$ plane ........................................... 92
5.2 An acceptability region for specification 1 on $x_7-x_{17}$ plane ........................................... 93
5.3 An acceptability region for specification 2 on $x_7-x_6$ plane ........................................... 93
5.4 An acceptability region for specification 2 on $x_7-x_{17}$ plane ........................................ 94
CHAPTER 1

INTRODUCTION

Recently, with the cost of the computer becoming increasingly affordable, the use of computers has become widespread. The demand for more complex microelectronic devices is hence greater than ever. To produce better devices, the semiconductor industry needs to advance the current deep sub-micron technology. The objective of the device design is to find semiconductor devices which are optimal with respect to some design goals on its electronic performances. Each design goal is a complex function of many parameters of manufacturing process and device structure. This chapter first gives a brief introduction on process flow and semiconductor devices in Section 1.1. We then present the device design optimization problem and the device decomposition problem in Section 1.2. The previous approach and related work are reviewed in Section 1.3. The contributions and the outline of the dissertation are presented in Section 1.4 and Section 1.5.

1.1. Semiconductor Device Fabrication Process

![Figure 1.1: IC process flow](image_url)

Figure 1.1: IC process flow
An integrated circuit manufacturing process flow is a sequence of precisely controlled fabrication steps, each performing a specific operation on the silicon wafer. A flow diagram of a fabrication process is shown in Figure 1.1. It is an iterative sequence with some major steps. Each major step can consist of one or more minor steps. Some typical major and minor steps are described in Table 1.1.

An example of a fabrication process flow for one of the simplest devices, a p-n junction, is demonstrated in Figure 1.2. In step 1, the silicon wafer is prepared. It contains a p-type material (e.g. boron). In step 2, the oxidation process grows a thin film of silicon dioxide. In step 3, a pattern of photo-resist is applied through a mask. In step 4, the etch process removes oxide on the uncovered area. In step 5, the n-type dopant is introduced into the silicon through the area not covered by oxide. A common n-type impurity element is phosphorus. In step 6, a thin layer of metal is deposited on the surface. This metal is used as a contact for the p-n junction. In step 7, a pattern of photo-resist is applied to form the area where the p-n diode is to be constructed. In step 8, the etch process removes the metal in the un-protected area. With the photo-resist removed in step 9, we have a p-n junction. The design of practically all semiconductor devices is based on the p-n junction.

With a little more complex sequence, we can have an NMOS transistor as shown in Figure 1.3. A MOS transistor is one of the semiconductor devices that most companies and much of the technology development effort has focused on. Each semiconductor device has its electronic properties. They are measured by a few performance indices. Some typical performance indices are the drive current of a transistor $I_{drive}$, the maximum allowable leakage current $I_{off}$, and the threshold voltage $V_t$. Each performance
Table 1.1: Some typical process steps

<table>
<thead>
<tr>
<th>Major Steps</th>
<th>Minor Steps</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Material Preparation</td>
<td>Purify silicon(Si)</td>
<td>provide high-quality single-crystal semiconductor wafer</td>
</tr>
<tr>
<td></td>
<td>Grow Si crystal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Prepare wafer</td>
<td></td>
</tr>
<tr>
<td>Deposit or Grow Film</td>
<td>Epitaxial growth</td>
<td>grow a thin single-crystal film</td>
</tr>
<tr>
<td></td>
<td>Oxide growth</td>
<td>grow oxide on exposed wafer surface</td>
</tr>
<tr>
<td></td>
<td>Chemical vapor deposition</td>
<td>deposits a thin film material by reacting chemical vapor on the wafer surface</td>
</tr>
<tr>
<td></td>
<td>Metalization</td>
<td>provides low resistivity metal area for IC interconnect, bonding pad and junction</td>
</tr>
<tr>
<td>Photolithography</td>
<td>Apply photo-resist expose, develop</td>
<td>put a pattern of photo-resist on the wafer to allow for selective processing of the surface</td>
</tr>
<tr>
<td>Etch Film</td>
<td>Wet etch</td>
<td>use liquid chemical to selectively remove material from wafer surface</td>
</tr>
<tr>
<td></td>
<td>Dry etch</td>
<td>use plasma energy driven reactive gas to remove material</td>
</tr>
<tr>
<td>Other Processing</td>
<td>Diffusion</td>
<td>diffuse dopant atoms from unmasked surface to modify electrical properties of a semiconductor material</td>
</tr>
<tr>
<td></td>
<td>Ion implantation</td>
<td>introduce dopant or create a dielectric layer by bombarding with a beam of dopant ions</td>
</tr>
<tr>
<td></td>
<td>Plasma processing</td>
<td>deposit films or dry etch</td>
</tr>
<tr>
<td></td>
<td>Any previous step</td>
<td></td>
</tr>
</tbody>
</table>
1. Material Preparation
2. Oxidation
3. Photolithography
4. Etch
5. Diffusion
6. Metalization
7. Photolithography
8. Etch

Figure 1.2: A p-n junction fabrication process sequence

Figure 1.3: A NMOS transistor
is determined by many device structure and doping parameters. Some typical examples are the length of the gate electrode $L_{gate}$, the thickness of the gate oxide $T_{ox}$, and the doping profile that describes the distribution of the impurity concentration. These device structure parameters and doping parameters are in turn determined by process control parameters such as time, temperature, and pressure. A simplified example of a performance model is:

$$I_{drive} = c_0 + c_1 * T_{ox} + c_2 * L_{gate} + c_3 * T_{ox}^2 + c_4 * L_{gate}^2 + c_5 * T_{ox} * L_{gate}$$

A real normalized 20-dimensional example from Texas Instruments is shown in Section 5.2. Depending on the need, the device performance models can be constructed in terms of (1) device structure and doping parameters [40] [41] [39], or (2) process parameters [4]. For simplicity, we use the term designable to represent both types of variables that are chosen to build the model.

1.2. Device Design Optimization

Given the device performance models, the designer specifies a range for each device performance index. These performance requirements that the device design must achieve are called the device performance specifications. The objective of the device optimization is to find an optimal design point according to an objective function in the designable space which meets all performance specifications. Often, there are multiple design criteria like minimizing $I_{off}$ or $I_{on}$. Different design interests can conflict with each other. One way to formulate an objective function in this situation is to form a weighted composite function of these design criteria.
One optimization method commonly followed is to use a nonlinear optimizer to iteratively search for an optimal device design point. Because there are many optimization algorithms, this optimizer based method is also widely adapted in many engineering disciplines [16] [3] [42] [35]. However, the nonlinear optimization problem is very difficult. In this method, a composite objective function is often used as an optimality criterion in the optimizer. The designer first prioritizes different design goals and assigns their weights. The composite objective function is simply the sum of the weighed design goals. A slightly different weight assignment can lead to a very different optimal design point when the constraint models are non-linear, which is generally true in practice. More importantly, the major drawback of this approach is that the global optimality of a solution is not guaranteed by the nonlinear optimization algorithms.

In this dissertation, we study a problem that computes the set of all feasible solutions, called acceptability region, such that every point inside the region meets all the performance specifications. This problem is also studied in the design environment [40] [41] [39] [37]. We call this problem device decomposition problem and define it in Section 1.2.1. We present a new and efficient representation for acceptability regions and a novel linearization technique for device models. Based on our acceptability region representation, we present an adaptive method for the device design optimization.

1.2.1. The Device Decomposition Problem

The device decomposition problem decomposes the device performance requirement into an acceptability region in the designable space. It is defined as follows,
Definition 1.1. Device Decomposition Problem

Given

- A set of device performance models, \( y_i = f_i(X), i = 1, 2, \ldots, m \), where \( y_i \) is a performance variable, \( X = [x_1, \ldots, x_n] \) is a vector of designable variables.

- A range for each designable, \( a_{ik} \leq x_k \leq a_{uk}, k = 1, \ldots, n \), within which the performance models are constructed.

Query:

- A set of designer specified performance specifications, \( c_{li} \leq y_i \leq c_{ui}, i = 1, 2, \ldots, m \).

Answer:

- An acceptability region \( AR = \{ X | c_{li} \leq f_i(X) \leq c_{ui}, i = 1, 2, \ldots, m \} \).

Note that for any problem that have a linear inequality \( q(Y) < c \) as a performance specification, one can convert it to the format in Definition 1.1 by the following two steps.

1. Convert \( q(Y) \) to \( q'(X) \) by plugging in \( y_i = f_i(X) \).

2. Introduce a new performance model \( y_{new} = q'(X) \) and a performance specification \( y_{new} < c \).

In this dissertation, we study the computation of an approximate acceptability region \( AR' \). To ensure its accuracy, the designer can set a bound on each error tolerance \( T_i \), such that for any point \( p \) in \( AR' \), \( c_{li} - T_i \leq f_i(p) \leq c_{ui} + T_i, i = 1, \ldots, m \).
One method [40] [41] [39] to obtain the device performance model is to construct response surface models [5]. The data for the response surface models can be obtained from the following two sources.

- **Numerical device simulator**: When the response surface models are obtained from a computationally expensive simulator, they can serve as efficient low-order polynomial replacements for the simulator.

- **Measurement of characterization experiments**: The response surface models can also be constructed by using the design of experiments (DOE) suitable for capturing linear, quadratic, and interaction terms.

In either case, low order is a good assumption because physical processes are adequately approximated by low order Taylor series in a small enough range. Because of the low-order nature of the models, we focus our work on quadratic device models in this dissertation.

An example of deriving an acceptability region is shown in Figure 1.4. In this example, the user first specifies an acceptable performance range in the domain of \( V_t \) and \( I_{off} \). Both performance models, \( V_t \) and \( I_{off} \), are a function of two designables, \( T_{ox} \) and \( L_{gate} \). The goal of device decomposition is to find the acceptability region in the domain of \( T_{ox} \) and \( L_{gate} \) so that the performances of all design points in the acceptability region satisfy the performance requirement.
1.3. Related Work

A Previous Approach

The device decomposition problem is first raised by Saxena, et al [40] [41] [39]. An algorithm is also provided for the device decomposition problem. It consists of the following two steps.

1. For each designable variable $x_i$, $i = 1, \ldots, n$, a sensitivity study is performed for each model $y_j$, $j = 1, \ldots, m$, at a point $p = (p_1, p_2, \ldots, p_n)$. It computes a maximal $\Delta x_i$ so that

$$\frac{1}{2} \left[ \max_{p_i - \frac{\Delta x_i}{2} \leq x_i \leq p_i + \frac{\Delta x_i}{2}} y_j(p_1, \ldots, p_{i-1}, x_i, p_{i+1}, \ldots, p_n) - \min_{p_i - \frac{\Delta x_i}{2} \leq x_i \leq p_i + \frac{\Delta x_i}{2}} y_j(p_1, \ldots, p_{i-1}, x_i, p_{i+1}, \ldots, p_n) \right] \leq \frac{T_j}{n},$$

where $T_j$ is the error tolerance for the model $y_j$. The point $p$ is often the center of the whole design space.

2. The designable space is then partitioned into a set of hyperboxes by dividing each coordinate $x_i$ with $\Delta x_i$. Each hyperbox is then evaluated at its center point to determine if every predicted device performance in this hyperbox meets the
requirements. The acceptability region in designable space is then represented by the set of qualified hyperboxes.

Because the sensitivity studies are performed for one designable variable at a time, the actual error is the combination of the errors from all designable variables. To ensure the actual error is less than the error tolerance, each maximum error tolerance \( T_j \) is divided by \( n \) in step 1. In addition to the time needed for sensitivity studies, the time complexity for step 2 is \( \Theta(q^n \times C) \), where \( q \) is the number of subdivisions on each coordinate, and \( C \) is the time to evaluate all performance models. This method has following drawbacks;

1. Since all sensitivity studies are done at the point \( p \), the accuracy checking condition in step 1 is not globally true. Consequently, this representation is not globally accurate.

2. The second step needs to be computed every time the designer specifies a new requirement. Because it is exhaustive, its complexity depends on the total number of hyperboxes in the designable space. As a result, because of the combinatorial explosion nature of the designable space partitioning, experiments have shown this approach handles only up to 6 to 8 dimensions for typical device decomposition problems.

Related Work in Artificial Intelligence

The problem of device decomposition can be viewed as an instance of the constraint satisfaction problem in artificial intelligence (AI). A constraint satisfaction problem (CSP) consists of a set of variables \( x_i, i = 1, \ldots, n \), each of which has an associated domain \( D_i \), and a set of constraints. A constraint \( c \) on variables \( x_{i_1}, \ldots, x_{i_j} \) is a sub-
set of the Cartesian product $D_{i_1} \times \cdots \times D_{i_j}$. A solution to a CSP is an assignment of values to all variables in such a way that all constraints are satisfied. A large number of problems in many areas of computer science can be viewed as special cases of CSP [27]. Some examples are scheduling [18] [34] [36], belief maintenance [13], temporal reasoning [12], machine design and manufacturing [30], and floor plan design [15].

A great number of experiments and techniques had been studied in the literature on CSPs. We can identify them by two factors. The first factor is the type of the variable domain. Some techniques study the problem whose variable domains are discrete and finite. The others are for the problem where the domains of variables are continuous intervals. Most of the researches focus on the problem with finite discrete variable domain. This set of researches can be further divided by the second factor, the completeness of the solution. One set deals with the problem in which finding one solution is sufficient, the other is for the problem in which finding all solutions is necessary. All problems in NP like 3-satisfiability problem and graph K-colorability problem [21] are discrete-finite-domain single-solution CSPs.

The backtracking paradigm has been the major method to find solutions. It sequentially finds a feasible value assignment for each variable. At any point, if no feasible value can be found for the current variable, it backtracks to re-assign a feasible value for the previous variable. However, in the worst case, exponential time is needed because the search space is exponentially large. Many methods have been proposed to reduce the search space. Some techniques examine a subset of variables at a time and remove those value assignments which do not satisfy the related constraints [29] [22] [10]. In this case, searching for a solution may still need backtracking. The
other studies are dedicated to reduce the search space so that the search is backtrack-free [11] [46] [9].

Our problem requires the complete set of solutions in the continuous variable domain. Research in this area is very limited. One recent and important result is a consistency technique by Sam-Harould [37]. It consists of the following three steps:

1. Each model $y_i$ is transformed into a set of ternary functions by repeating the following two operations;

   Replace each sub-expression $x_p \text{ op } x_q$ in $y_i$ with a new variable $x_{\text{new}}$.

   Introduce a new function $x_{\text{new}} = x_p \text{ op } x_q$.

2. For any three variables $x_i$, $x_j$, and $x_k$, the three-dimensional space $D_i \times D_j \times D_k$ is partitioned into a set of rectangular boxes. Each box is then examined to check if its region completely satisfies all ternary constraints on $x_i$, $x_j$, and $x_k$. Each ternary relation $R_{i,j,k}$ is represented by these identified feasible boxes. However, if the constraint is an equality, no feasible box will be identified. A relaxed representation is proposed for this situation. Each equality $x_i = f(x_j, x_k)$ is replaced by two inequalities, $x_i \geq f(x_j, x_k) - \frac{\delta}{2}$ and $x_i \leq f(x_j, x_k) + \frac{\delta}{2}$. Because of this, spurious solutions are included in this representation.

3. If all ternary relations are convex, the procedure 3-2.relational.consistency() is used to ensure 3-2 relational consistency on all relations. Three ternary relations $R_{i,u,v}$, $R_{j,u,v}$, and $R_{k,u,v}$ are 3-2 relational consistent relative to variable $x_u$ and $x_v$ if and only if any consistent instantiation of variables $x_i$, $x_j$, and $x_k$ has an extension to variables $x_u$ and $x_v$ that satisfies $R_{i,u,v}$, $R_{j,u,v}$, and $R_{k,u,v}$ simultaneously.
Procedure 3-2\_relational\_consistency() (by Sam-Harould et al.)

begin

\(\text{repeat}\)
\[\text{changed} = \text{false};\]
\[\text{foreach} \text{variable pair } x_u \text{ and } x_v,\]
\[\text{foreach} \text{variable triplet } x_i, x_j, \text{and } x_k,\]
\[R_{i,j,k}' = R_{i,j,k} \oplus \Pi_{i,j,k} R_{i,u,v} \otimes R_{j,u,v} \otimes R_{k,u,v}.\]
\[\text{if } R_{i,j,k}' \neq R_{i,j,k} \]
\[R_{i,j,k} = R_{i,j,k}';\]
\[\text{changed} = \text{true};\]
\[\text{until} \text{changed} == \text{false}\]
end

In the procedure 3-2\_relational\_consistency(), the operation \(R_1 \oplus R_2\) computes the intersection of \(R_1\) and \(R_2\). The operation \(\Pi_{i,j,k} R_1\) projects \(R_1\) into the space \(D_i \times D_j \times D_k\). The operation \(R_{i,u,v} \otimes R_{j,u,v}\) first extends \(R_{i,u,v}\) to \(R_{i,j,u,v}' = \{(p_i, p_j, p_u, p_v) \mid (p_i, p_j, p_u, p_v) \in D_i \times D_j \times D_u \times D_v,\) and \((p_i, p_u, p_v)\) satisfies \(R_{i,u,v}\}\) and extends \(R_{j,u,v}\) to \(R_{i,j,u,v}' = \{(p_i, p_j, p_u, p_v) \mid (p_i, p_j, p_u, p_v) \in D_i \times D_j \times D_u \times D_v,\) and \((p_j, p_u, p_v)\) satisfies \(R_{j,u,v}\}\), then returns \(R_{i,j,u,v}' \oplus R_{i,j,u,v}''\).

Sam-Harould and Faltings prove that for a set of convex relations, after 3-2 relational consistency checking, one can iteratively find a feasible value assignment from the relations for each variable without backtracking. The solution found will satisfy all relations. The set of relations with the backtracking-free property is called \textit{global}
consistent. Hence, we can use the set of $O(N^3)$ ternary relations to represent the acceptability region, where $N$ is the number of both original and newly introduced variables. The complexity of the procedure $3$-$2$-relational consistency() is $O(N^8q^8)$, where $q$ is the number of subdivisions of each coordinate. This polynomial complexity is a great improvement over the previous exponential one by Saxena et al. However, the situation without the convexity property remains NP-Hard, which is shown in the following theorem.

**Theorem 1.1.** Computing global consistency for a set of ternary relations constructed by Sam-Harould's method for a continuous CSP is NP-Hard.

**Proof:**

This proof is done by a reduction from 3SAT. Let $S = (U, C)$ be an instance of 3SAT problem, where $U = \{u_1, u_2, \ldots, u_n\}$ is a set of boolean variables and $C = \{c_1, c_2, \ldots, c_m\}$ is a set of clauses. We construct a global consistency problem for a set of ternary relations as follows: For each $u_i$, we have a variable $x_i$ which takes values from two intervals $I_f = [0, 0]$ and $I_t = [1, 1]$. The variable $u_i$ being false corresponds to $x_i$ taking a value from $I_f$ and the variable $u_i$ being true corresponds to $x_i$ taking a value from $I_t$. We construct $\binom{n}{3}$ relations $R_{i,j,k} \subseteq \{I_f, I_t\}^3$ for every three variables $x_i, x_j, \text{and } x_k$, $1 \leq i, j, k \leq n, i \neq j \neq k$. For each clause $c$ containing $u_i, u_j$ and $u_k$, we delete a solution space subdivision from the relation $R_{i,j,k}$ so that the variable $x_i, x_j, \text{and } x_k$ can not take values from it. This space subdivision corresponds to the value assignments for $u_i, u_j \text{ and } u_k$ that will make the clause $c$ false. An example is shown in Figure 1.5. The overall time complexity of the construction process is $O(n^3)$.

Any value assignment for $x_i, x_j, \text{and } x_k$ in $R_{i,j,k}$ will guarantee a value assignment for $u_i, u_j \text{ and } u_k$ to make the clause $c$ true. Consequently, there exists an interval for
each $x_i, 1 \leq i \leq n$ that satisfies all relations if and only if $S$ is satisfiable. Identifying all feasible solutions to achieve global consistency is hence NP-hard.

Figure 1.5: An example of ternary relation construction from a clause

The problem for using this method for device decomposition can be summarized as follows:

1. The combination of the ternary function transformation and the relaxed equality representation can introduce inaccuracy. For example, after transformation, a linear model $y = ax_1 + bx_2 + cx_3 + dx_4$ becomes,

$$
y = t_1 + t_2;
$$

$$
t_1 = ax_1 + bx_2;
$$

$$
t_2 = cx_3 + dx_4; \quad (1.1)
$$

By relaxed representation, we have,

$$
t_1 + t_2 - \frac{\varepsilon_1}{2} \leq y \leq t_1 + t_2 + \frac{\varepsilon_1}{2};
$$
\[
ax_1 + bx_2 - \frac{\varepsilon_2}{2} \leq t_1 \leq ax_1 + bx_2 + \frac{\varepsilon_2}{2}; \\
cx_3 + dx_4 - \frac{\varepsilon_3}{2} \leq t_2 \leq cx_3 + dx_4 + \frac{\varepsilon_3}{2};
\]

(1.2)

Equivalently, the original model \( y \) is represented by

\[
ax_1 + bx_2 + cx_3 + dx_4 - \frac{\varepsilon_1 + \varepsilon_2 + \varepsilon_3}{2} \leq y \leq ax_1 + bx_2 + cx_3 + dx_4 + \frac{\varepsilon_1 + \varepsilon_2 + \varepsilon_3}{2}
\]

(1.3)

The resulting relaxed representation can greatly increase the solution space of the original model function. In general, the relaxed representation of \( y \) becomes

\[ f(X) - \Phi_1(X, \varepsilon_1, \varepsilon_2, \ldots) \leq y \leq f(X) + \Phi_2(X, \varepsilon_1, \varepsilon_2, \ldots). \]

The more complex the model function is, the greater effects the \( \Phi_1 \) and \( \Phi_2 \) have. Consequently, the resulting acceptability region can be greatly enlarged. A one-dimensional example demonstrating this situation is shown in Figure 1.6.

2. When the convexity condition is not true, the problem is NP-hard. In order to apply the 3-2 relational consistency technique, each non-convex relation needs
to be decomposed into a set of convex ones, then the technique can be applied to every combination of convex sub-relations from each decomposed non-convex relation. Its complexity becomes $O(k^{N^3} N^{8} d^8)$, where $k$ is the number of decomposed convex sub-relations for each non-convex relation.

Related Work from Design Centering

The process variation in circuit manufacturing causes the design parameters to change from its nominal values. Hence, the behavior of the circuit may not meet the required design specifications. The problem of design centering is to select the nominal values of design parameters to ensure that, with the greatest probability, the behavior of the circuit remains within specifications. As a result, the manufacturing yield is maximized.

One type of approaches, often referred as geometrical approaches, consist of the following two steps.

1. Computes a geometrical body to approximate the feasible region in the space of design parameters, i.e., the region where the behavioral specifications are satisfied.

2. Find an approximate center point of the computed geometrical body as the design center.

One approach proposed to use an ellipsoid to approximate the feasible region and find a design center within the ellipsoid [1]. Other approaches proposed to first use a convex polytope to approximate the feasible region and inscribe the largest hypersphere [14] or Hessian ellipsoid [38] within the convex polytope, then find a
design center within the hypersphere or the ellipsoid. The feasible region computation in the step 1 is very similar to the acceptability region computation for the device decomposition problem. But the objective of the design centering is to find a good design center, not an accurate feasible region representation. These approaches can assume the feasible region to be convex, which is not valid for acceptability region for device decomposition problems. Hence, the feasible region representation for design centering is not suitable for the acceptability region computation.

Summary of Existing Methods

Because of the shortcomings of the existing methods, we identify the following two improvement directions for our research;

- More efficient methods are needed in order to handle more complex and higher-dimensional problems.
- Better accuracy of the acceptability region representation is needed so that the maximum error of the computed acceptability region is within the specified tolerance.

1.4. Contributions

The work in this dissertation makes contributions in two areas.

1. The first area is the constraint satisfaction studied in artificial intelligence. Firstly, we propose the piecewise linearization concept for constraints and present a novel adaptive linearization technique for quadratic constraints. Secondly, we use a new polytope-based representation to represent acceptability regions. Because of the ability to control the accuracy of the piecewise linear approximation
of performance models, the computed acceptability region is globally accurate. Because of the efficiency of the polytope-based representation, we greatly reduce the intractable complexity encountered by the previous approaches.

2. The second area is semiconductor device design optimization. Based on our acceptability region representation, we propose an adaptive optimization design paradigm. It provides more information for exploring the design space and more flexibility to search for a globally optimal design point.

1.5. Outline of the Dissertation

- Chapter 2: This chapter describes the piecewise model linearization. A linearization algorithm for an arbitrary model is first presented. It then describes a novel adaptive linearization technique to approximate quadratic performance models accurately by piecewise linear functions.

- Chapter 3: The first half of this chapter presents a new convex polytope based representation for the acceptability region and shows how to derive the acceptability region, based on the piecewise linear approximations of performance models. The second half of this chapter demonstrates a method to display multi-dimensional acceptability regions in the new representation on a two-dimensional plane. The previous works on two-dimensional projection of convex polytope are reviewed. A new projection algorithm specifically for the two-dimensional case is presented.

- Chapter 4: Based on the acceptability region and the piecewise linearization technique, we present an adaptive design paradigm to study device design and compute a globally optimal solution.
• Chapter 5: This chapter presents an implementation of our algorithms and their experimental results.

• Chapter 6: This chapter summarizes this dissertation and concludes with directions for future research.
CHAPTER 2

MODEL LINEARIZATION AND REPRESENTATION

This chapter presents a new and efficient representation for device performance models: the piecewise linear function representation. It is the first step and the foundation of our approach. Its efficiency in handling high-dimensional device decomposition problems is the major advantage for our approach. Compared to the previous representations,

1. It is more efficient because the linear function can produce a better fit than the piecewise constant function for the curvature of the device models. As a result, less number of pieces are needed.

2. It is more accurate because the linearization error is globally controlled in the linearization process.

3. It can be generated online depending on any representation error tolerance without any sensitivity study.

In Section 2.1, we discuss the reason of the major shortcomings of the piecewise constant representation used in both the previous approach [40] [41] [39] and the constraint satisfaction problem within artificial intelligence community [37]. We then introduce the concept of the piecewise linear representation that will greatly reduce the drawback of the piecewise constant representation. Section 2.2 to Section 2.4 give details about our adaptive piecewise linearization technique for quadratic device performance models. In Section 2.5, we discuss a data structure, the binary space partitioning (BSP) tree, for the piecewise linear representation.
2.1. From Piecewise Constant to Piecewise Linear

To represent a model function \( y = f(x_1, x_2, \ldots, x_n) \), the piecewise constant approach consists of the following two steps.

1. Divide the space of all designables involved in the model function, \( y, x_1, x_2, \ldots, x_n \), into rectangular subdomains.

2. Determine for each rectangular subdomain whether it is a part of the acceptability region representation.

Because of the accuracy requirements, each subdomain must be small enough so that the error in each subdomain is less than the specified error tolerance. The drawback of the piecewise constant representation is the number of its subdomains. Because of the approximation limitation of the constant function, the domain of each designable’s variable needs to be divided into a great many of segments. The complexity of this representation is the multiplication of the number of the divisions within the domain of each variable. It blows up very quickly and puts a major limitation on the dimensionality of the problem that can be handled.

On the other hand, our linearization technique consists of the following two parts.

1. Divide the space of all designable variables, \( x_1, x_2, \ldots, x_n \), into a set of rectangular designable space subdivisions.

2. Compute a linear function to approximate the model function within each subdivision, so that the error between the linear approximation and the original function is minimal.
Piecewise linear representation relies on the fact that a linear function can approximate the model function much better than a constant function. The number of the subdivisions for each designable variable is much less. A one dimensional example is shown in Figure 2.1. In the piecewise constant representation in the figure, the error grows much faster as the value of the input variable grows larger. As a result, the size of an input subdivision needs to be very small to preserve the accuracy. On the other hand, the linear function is much more powerful. Only two subdivisions are needed in the example. As the dimensionality increases, the number of designable space partitions reduced by the piecewise linear representation is even greater.

![Figure 2.1: Comparison of piecewise constant and piecewise linear representation](image)

An example of constructing a piecewise linear representation is shown in Figure 2.2. The space of designable variables is partitioned into fours subdivisions. In each subdivision, a linear approximation function is computed to fit the original model. Since the resulting piecewise linear model is constructed based on a space partitioning, we choose the binary space partitioning tree as its data structure. It keeps all pieces of linear functions in a tree structure according to its location. An example is in Figure 2.3. The concept of this tree data structure will be introduced in Section 2.5.
Figure 2.2: Adaptive Piecewise Linearization

Figure 2.3: The data structure for the piecewise linear model
An algorithm to linearize models without any function type assumption is presented in the procedure General-Piecewise-Linear(). It recursively partitions the input space and computes linear approximations until the maximum errors of all linear approximations are less than the error tolerance. This algorithm is general and simple. But it needs two expensive algorithms: the singular-value decomposition [25], to solve a linear least-square data fitting problem, and the down-hill simplex [33], to solve a nonlinear optimization problem. Hence, the dimension of problems it can handle is also expected to be very limited.

**Procedure General-Piecewise-Linear(domain d)**

```
begin
1 Select a set of random samples \( s \), from the model \( m \);
2 Use a linear least-square data fitting method, singular-value decomposition, to find a linear function \( l \) for \( s \);
3 Use a nonlinear optimization algorithm, down-hill simplex, to estimate the maximum linearization approximation error, \( e = \max( | m - l | ) \), within \( d \);
4 if \( e > \) the error tolerance
5 divide \( d \) into two halves \( d_1 \) and \( d_2 \);
6 General-Piecewise-Linear(\( d_1 \));
7 General-Piecewise-Linear(\( d_2 \));
8 else
9 store \( l \);
10 return;
end
```
A representation called Canonical Piecewise Linear Representation was introduced by Chua and Kang for the modeling and analysis of nonlinear devices, circuits and systems [8] [26]. Its definition is as follows,

\[ \bar{f}(x) = a + b'x + \sum_{i=1}^{s} c_i | \alpha_i' - \beta_i | \]  

(2.1)

where \( a, \beta_i, c_i \in R, b, \alpha_i \in R^n \), are the parameters of the structure. It is a non-linear combination of linear affine \( n \)-dimensional hyperplanes. Julián, Jordán and Desages present a recursive method, using a nonlinear optimization algorithm, to obtain simultaneously all the parameters of the representation [24]. They also present an iterative method to continue to partition the domain and refine the canonical piecewise linear representation. A three-dimensional example is implemented. But because of the difficulty of nonlinear optimization problem, it is not clear that this approach is feasible for problems of eight or more dimensions.

The remaining of this chapter will present a novel linearization technique for quadratic models. The technique to obtain the piecewise linear representation can be divided by answering the following four subproblems.

1. How do we find the linear function to approximate each model in a designable subdomain \( x_{i0} \leq x_i \leq x_{i0} + \Delta x, i = 1, 2, \ldots, n \) ?

2. What is the error between the linear approximation and the original model function within each designable subdomain?

3. How can the linearization error be used to find a designable space partitioning so that the accuracy tolerance is satisfied?
4. How can the piecewise linear representation in the binary space partitioning form be determined, if the performance specification is given?

In the next three sections, we present a linearization method for the quadratic performance model. It is described by solving the above four subproblems. We demonstrate an analytic technique for question one in Section 2.2. The question two is discussed in Section 2.3. In this section, we also present a greedy algorithm to find a designable space partitioning, so that the error of each partition's linear approximation satisfies accuracy tolerance. The solution for question four is presented in Section 2.4.

2.2. Linear Approximation for a Quadratic Function

Our method of finding a linear function to approximate a quadratic performance model within \(x_{i0} \leq x_i \leq x_{i0} + \Delta x, i = 1, 2, \ldots, n\), is from studying the formulation of a quadratic function. We break the problem of linearizing a quadratic function \(y\) into a set of subproblems. Each subproblem studies the linearization of a functional term \(t_k, k = 0, 1, \ldots, r\). Once each subproblem is solved, we combine their solutions. We start by identifying the basic terms of a quadratic function. Each quadratic model function, \(a + \sum_{i=1}^{n} b_i x_i + \sum_{i,j=1}^{n} c_{ij} x_i x_j\), contains three types of terms, constant, linear, and quadratic terms. Since the first two types are already linear, we only need to study quadratic terms. A quadratic term is either a square or product term, i.e., \(x_i^2\) or \(x_i x_j\). We first demonstrate the technique for linearizing a square term and a product term, then present a simple method to obtain a linear approximation for a quadratic function within a subdomain.
2.2.1. Linearizing a square term $x_i^2$

In this section, we derive the method that determines the best linear function, $L_{A_{x_i^2}}(x_i)$, to approximate the term $x_i^2$ over the range $x_{i_0} \leq x_i \leq x_{i_0} + \Delta x_i$. In other words, the maximum difference between $L_{A_{x_i^2}}(x_i)$ and $x_i^2$ will be minimized.

Let the linear function $L(x_i)$ be the straight line between two points $p_1 = (x_{i_0}, x_{i_0}^2)$ and $p_2 = (x_{i_0} + \Delta x_i, (x_{i_0} + \Delta x_i)^2)$ on the function $f(x_i) = x_i^2$, we have

$$
L(x_i) = x_{i_0}^2 + \frac{(x_{i_0} + \Delta x_i)^2 - x_{i_0}^2}{\Delta x_i} (x_i - x_{i_0})
= x_{i_0}^2 + (2x_{i_0} + \Delta x_i)(x_i - x_{i_0}).
$$

(2.2)

We use the linear function $L(x_i)$ as an initial try-out and observe the following analysis on its approximation error. The difference between $L(x_i)$ and $x_i^2$ is,

$$
L(x_i) - x_i^2 = x_{i_0}^2 + (2x_{i_0} + \Delta x_i)(x_i - x_{i_0}) - x_i^2
= x_{i_0}^2 - (2x_{i_0} + \Delta x_i)x_{i_0} + (2x_{i_0} + \Delta x_i)x_i - x_i^2
= -x_{i_0}^2 - x_{i_0}\Delta x_i + 2x_i(x_{i_0} + \frac{\Delta x_i}{2}) - x_i^2
= \frac{\Delta x_i^2}{4} - (x_{i_0} + \frac{\Delta x_i}{2})^2 + 2(x_{i_0} + \frac{\Delta x_i}{2})x_i - x_i^2
= \frac{\Delta x_i^2}{4} - (x_{i_0} + \frac{\Delta x_i}{2} - x_i)^2.
$$

Since $0 \leq (x_{i_0} + \frac{\Delta x_i}{2} - x_i)^2 \leq \frac{1}{4}\Delta x_i^2$, for $x_{i_0} \leq x_i \leq x_{i_0} + \Delta x_i$,

$$
0 \leq L(x_i) - x_i^2 \leq \frac{1}{4}\Delta x_i^2 = \xi.
$$
Because of the above analysis, we define the linear approximation $L_Ax_i^2(x_i)$ as the function $L(x_i)$ translated by $\xi = \frac{1}{8} \Delta x_i^2$ and is derived as follows. An illustration is shown in Figure 2.4.

$$L_Ax_i^2(x_i) = L(x_i) - \frac{\xi}{2} = x_i^2 + (2x_i + \Delta x_i)(x_i - x_i) - \frac{1}{8} \Delta x_i^2.$$  \hspace{1cm} (2.3)

In the following two theorems, we first calculate the maximum linearization error of $L_Ax_i^2(x_i)$, $\text{max}(|L_Ax_i^2(x_i) - x_i^2|)$, within $x_i \leq x_i \leq x_i + \Delta x_i$, then we prove that it is optimal.

**Theorem 2.1.** The maximum linearization error of $L_Ax_i^2(x_i)$ over $x_i^2$ within the range $x_i \leq x_i \leq x_i + \Delta x_i$ is $\frac{1}{8} \Delta x_i^2$. 
Proof:

The linearization error of $L_{A2^2}(x_i)$,

$$L_{A2^2}(x_i) - x_i^2$$

$$= x_{i0}^2 + (2x_{i0} + \Delta x_i)(x_i - x_{i0}) - \frac{1}{8}\Delta x_i^2 - x_i^2$$

$$= \frac{1}{8}\Delta x_i^2 - (x_{i0} + \frac{\Delta x_i}{2} - x_i)^2.$$

Since $0 \leq (x_{i0} + \frac{\Delta x_i}{2} - x_i)^2 \leq \frac{1}{4}\Delta x_i^2$, for $x_{i0} \leq x_i \leq x_{i0} + \Delta x_i$.

$$-\frac{1}{8}\Delta x_i^2 \leq \frac{1}{8}\Delta x_i^2 - (x_{i0} + \frac{\Delta x_i}{2} - x_i)^2 \leq \frac{1}{8}\Delta x_i^2$$

$$|L_{A2^2}(x_i) - x_i^2| \leq \frac{1}{8}\Delta x_i^2$$

\(\square\)

**Theorem 2.2.** The linear function $L_{A2^2}(x_i)$ is the optimal linear approximation for $x_i^2$, where $x_{i0} \leq x_i \leq x_{i0} + \Delta x_i$.

**Proof:**

Let the function $y = ax_i + b$ be the optimal linear approximation of $x_i^2$ for $x_{i0} \leq x_i \leq x_{i0} + \Delta x_i$. The function $E(x_i) = x_i^2 - y = x_i^2 - ax_i - b$ is the corresponding linearization error function between $y$ and $x_i^2$. Since the linear approximation $y$ is optimal, we have

$$\max_{x_{i0} \leq x_i \leq x_{i0} + \Delta x_i} E(x_i) = -\min_{x_{i0} \leq x_i \leq x_{i0} + \Delta x_i} E(x_i).$$
Let
\[ E_{\text{range}} = \max_{x_i \leq x_i \leq x_{i0} + \Delta x_i} E(x_i) - \min_{x_i \leq x_i \leq x_{i0} + \Delta x_i} E(x_i). \]

The maximum error for \( y \) is \( \frac{E_{\text{range}}}{2} \). The parabola \( E(x_i) \) has the minimum value \( \frac{a^2}{4} - b \) at point \( x_i = \frac{a}{2} \). This point can be in three cases, \( \frac{a}{2} < x_{i0}, x_{i0} \leq \frac{a}{2} \leq x_{i0} + \Delta x_i, \) and \( x_{i0} + \Delta x_i < \frac{a}{2} \). The maximum error is analyzed in each case.

Case 1: \( \frac{a}{2} < x_{i0} \)

Since \( E(x_i) \) monotonically increases from \( x_{i0} \) to \( x_{i0} + \Delta x_i \), the maximum error,
\[
\max_{x_{i0} \leq x_i \leq x_{i0} + \Delta x_i} (|E(x_i)|) = \frac{E_{\text{range}}}{2}
= \frac{E(x_i)|_{x_i = x_{i0} + \Delta x_i} - E(x_i)|_{x_i = x_{i0}}}{2}
= \frac{1}{2}[(x_{i0} + \Delta x_i)^2 - a(x_{i0} + \Delta x_i) - b - x_{i0}^2 + ax_{i0} + b]
= \frac{1}{2}[\Delta x_i^2 + \Delta x_i(2x_{i0} - a)].
\]

Since \( x_{i0} > \frac{a}{2} \),
\[
\max_{x_{i0} \leq x_i \leq x_{i0} + \Delta x_i} (|E(x_i)|) > \frac{\Delta x_i^2}{2}.
\]

Case 2: \( \frac{a}{2} > x_{i0} + \Delta x_i \)

Since \( E(x_i) \) monotonically increases from \( x_{i0} + \Delta x_i \) to \( x_{i0} \), the maximum error,
\[
\max_{x_{i0} \leq x_i \leq x_{i0} + \Delta x_i} (|E(x_i)|) = \frac{E_{\text{range}}}{2}
= \frac{E(x_i)|_{x_i = x_{i0}} - E(x_i)|_{x_i = x_{i0} + \Delta x_i}}{2}
= \frac{1}{2}[x_{i0}^2 - ax_{i0} - b - (x_{i0} + \Delta x_i)^2 + a(x_{i0} + \Delta x_i) + b]
= \frac{1}{2}[-\Delta x_i^2 - \Delta x_i(2x_{i0} - a)].
\]

Since \( x_{i0} + \Delta x_i < \frac{a}{2} \)
\[
\max_{x_{i0} \leq x_i \leq x_{i0} + \Delta x_i} (|E(x_i)|) > \frac{\Delta x_i^2}{2}.
\]
Case 3: $x_{i0} \leq \frac{a}{2} \leq x_{i0} + \Delta x_i$

Without lost of generality, we assume $x_{i0} + \Delta x_i - \frac{a}{2} > \frac{a}{2} - x_{i0}$, that is, $\frac{\Delta x_i}{2} \leq x_{i0} + \Delta x_i - \frac{a}{2} \leq \Delta x_i$.

$$\max_{x_{i0} \leq x_i \leq x_{i0} + \Delta x_i} \left( |E(x_i)| \right) = \frac{E_{\text{range}}}{2}$$

$$= \frac{E(x)_{x_i=x_{i0}+\Delta x_i} - E(x)_{x_i=x_{i0}}}{2}$$

$$= \frac{1}{2} [(x_{i0} + \Delta x_i)^2 - a(x_{i0} + \Delta x_i) - b - (\frac{a}{2})^2 + a(\frac{a}{2}) + b]$$

$$= \frac{1}{2} (x_{i0} + \Delta x_i - \frac{a}{2})^2$$

$$\geq \frac{1}{2} (\frac{\Delta x_i}{2})^2$$

$$= \frac{\Delta x_i^2}{8}.$$ 

From the above, we know that the minimum of the maximum error of any linear approximation is $\frac{\Delta x_i^2}{8}$, which is the maximum error of $LA_{x_i^2}(x_i)$. Hence, $LA_{x_i^2}(x_i)$ is optimal.

\[\Box\]

With similar arguments, we have the following property for the linear approximation of $ax_i^2$, $LA_{ax_i^2}(x_i)$.

**Corollary 2.1.**  (1) $LA_{ax_i^2}(x_i) = a \cdot LA_{x_i^2}(x_i)$.

(2) The linear function $a \cdot LA_{x_i^2}(x_i)$ is the optimal linear approximation for the function $ax_i^2$, where $x_{i0} \leq x_i \leq x_{i0} + \Delta x_i$.

2.2.2. Linearizing a product term $x_ix_j$

In this section, we derive the method which determines the best linear function, $LA_{x_ix_j}(x_i, x_j)$, to approximate the function $x_ix_j$ over the domain $x_{i0} \leq x_i \leq x_{i0} + \Delta x_i$.
and \( x_{j0} \leq x_j \leq x_{j0} + \Delta x_j \).

Since

\[
x_i x_j = \frac{1}{2}[(x_i + x_j)^2 - x_i^2 - x_j^2],
\]

(2.4)
we can substitute each term \((x_i + x_j)^2\), \(x_i^2\), and \(x_j^2\), with a linear function and form a composite linear function to approximate the function \(x_i x_j\). For the term \(x_i^2\), we replace it with

\[
L_1(x_i) = x_{i0}^2 + (2x_{i0}^2 + \Delta x_i)(x_i - x_{i0}).
\]

The function \(L_1(x_i)\) is the straight line that pass through two points \((x_{i0}, x_{i0}^2)\) and \((x_{i0} + \Delta x_i, (x_{i0} + \Delta x_i)^2)\) on the function \(f(x_i) = x_i^2\). Similarly, we have

\[
L_2(x_j) = x_{j0}^2 + (2x_{j0}^2 + \Delta x_j)(x_j - x_{j0})
\]

for the term \(x_j^2\). We can also apply the same method on the term \((x_i + x_j)^2\) with the following transformation:

\[
z = x_i + x_j,
\]

\[
z_0 \leq z \leq z_0 + \Delta z,
\]

\[
z_0 = x_{i0} + x_{j0},
\]

\[
\Delta z = \Delta x_i + \Delta x_j.
\]

For the term \(z^2\), we have

\[
L_3(z) = z_0^2 + (2z_0^2 + \Delta z)(z - z_0).
\]

By substituting \(L_1(x_i)\), \(L_2(x_j)\), and \(L_3(z)\) to function 2.4, we have a initial try-out linear approximation function \(L_0(x_i, x_j)\) for \(x_i x_j\) as follows,
\[ L_0(x_i, x_j) \]
\[ = \frac{1}{2} [L_3(z) - L_1(x_i) - L_2(x_j)] \]
\[ = \frac{1}{2} \left[ (x_{i0} + x_{j0})^2 + \frac{(x_{i0} + x_{j0} + \Delta x_i + \Delta x_j)^2 - (x_{i0} + x_{j0})^2}{\Delta x_i + \Delta x_j} (x_i - x_j - \Delta x_i - \Delta x_j) \right. \]
\[ - \left. [x_{i0}^2 + (2x_{i0}^2 + \Delta x_i)(x_i - x_{i0})] - [x_{j0}^2 + (2x_{j0}^2 + \Delta x_j)(x_j - x_{j0})] \right] \}
\[ = \frac{1}{2} \left\{ 2x_{i0}x_{j0} + [2(x_{i0} + x_{j0}) + (\Delta x_i + \Delta x_j)](x_i - x_{i0} + x_j - x_{j0}) \right. \]
\[ - (2x_{i0} + \Delta x_i)(x_i - x_{i0}) - (2x_{j0} + \Delta x_j)(x_j - x_{j0}) \}
\[ = \frac{1}{2} \left\{ 2x_{i0}x_{j0} + (2x_{i0} + \Delta x_i)(x_j - x_{j0}) + (2x_{j0} + \Delta x_j)(x_i - x_{i0}) \right\} \]
\[ = x_{i0}x_{j0} + (x_{i0} + \frac{\Delta x_i}{2})x_j + (x_{j0} + \frac{\Delta x_j}{2})x_i - (x_{i0} + \frac{\Delta x_i}{2})x_{j0} - (x_{j0} + \frac{\Delta x_j}{2})x_{i0} \]
\[ = \left( x_{i0} + \frac{\Delta x_i}{2} \right)x_j + \left( x_{j0} + \frac{\Delta x_j}{2} \right)x_i - x_{i0}x_{j0} - \frac{\Delta x_i x_{j0}}{2} - \frac{\Delta x_j x_{i0}}{2} \]
\[ = \left( x_{j0} + \frac{\Delta x_j}{2} \right)x_i + \left( x_{i0} + \frac{\Delta x_i}{2} \right)x_j - \frac{1}{2} \left[ (x_{i0} + \Delta x_i)x_{j0} + (x_{j0} + \Delta x_j)x_{i0} \right]. \]

We then analyze the approximation error of \( L_0(x_i, x_j) \) as follows,

\[ L_0(x_i, x_j) - x_ix_j \]
\[ = \frac{1}{2} [L_3(z) - L_1(x_i) - L_2(x_j)] - x_ix_j \]
\[ = \frac{1}{2} \left\{ L_3(z) - (x_i + x_j)^2 - [L_1(x_i) - x_i^2] - [L_2(x_j) - x_j^2] \right\} \]
\[ = \frac{1}{2} \left\{ \frac{1}{4} (\Delta x_i + \Delta x_j)^2 - (x_{i0} + x_{j0} + \frac{\Delta x_i}{2} + \frac{\Delta x_j}{2} - x_i - x_j)^2 \right. \]
\[ - \left. \frac{\Delta x_i^2}{4} + (x_{i0} + \frac{\Delta x_i}{2} - x_i)^2 - \frac{\Delta x_j^2}{4} + (x_{j0} + \frac{\Delta x_j}{2} - x_j)^2 \right\} \]
\[ = \frac{1}{2} \left\{ \frac{\Delta x_i \Delta x_j}{2} - 2(x_{i0} + \frac{\Delta x_i}{2} - x_i)(x_{j0} + \frac{\Delta x_j}{2} - x_j) \right\} \]

Since
\[ -\frac{\Delta x_i \Delta x_j}{2} \leq -2(x_{i0} + \frac{\Delta x_i}{2} - x_i)(x_{j0} + \frac{\Delta x_j}{2} - x_j) \leq \frac{\Delta x_i \Delta x_j}{2}. \]
therefore,

$$0 \leq \frac{\Delta x_i \Delta x_j}{2} - 2(x_{i0} + \frac{\Delta x_i}{2} - x_i)(x_{j0} + \frac{\Delta x_j}{2} - x_j) \leq \Delta x_i \Delta x_j,$$

$$0 \leq L_0(x_i, x_j) - x_i x_j \leq \frac{\Delta x_i \Delta x_j}{2} = \xi.$$

Figure 2.5: Linearization of the product term, $x_i x_j$

From the above analysis, we define the linear approximation function $L A_{x_i x_j}(x_i, x_j)$ as the function $L_0(x_i, x_j)$ translated by $\xi$. It is derived as follows. An illustration is in Figure 2.5.

$$L A_{x_i x_j}(x_i, x_j)$$

$$= L_0(x_i, x_j) - \frac{\xi}{2}$$

$$= (x_{j0} + \frac{\Delta x_j}{2})x_i + (x_{i0} + \frac{\Delta x_i}{2})x_j - \frac{1}{2}[(x_{i0} + \Delta x_i)x_{j0} + (x_{j0} + \Delta x_j)x_{i0}]$$

$$- \frac{\Delta x_i \Delta x_j}{4}.$$  (2.5)
In the following two theorems, we first derive the maximum linearization error of $L_{A_{x_i x_j}}(x_i, x_j)$, then prove that it is also optimal.

**Theorem 2.3.** The maximum linearization error of $L_{A_{x_i x_j}}(x_i, x_j)$ for $x_{i0} \leq x_i \leq x_{i0} + \Delta x_i, x_{j0} \leq x_j \leq x_{j0} + \Delta x_j$ is $\frac{1}{4} \Delta x_i \Delta x_j$.

**Proof**

The linearization error of $L_{A_{x_i x_j}}(x_i, x_j)$,

$$L_{A_{x_i x_j}}(x_i, x_j) - x_i x_j$$

$$= \left( x_{i0} + \frac{\Delta x_i}{2} \right) x_i + \left( x_{j0} + \frac{\Delta x_j}{2} \right) x_j - \frac{1}{2} \left[ (x_{i0} + \Delta x_i)x_{j0} + (x_{j0} + \Delta x_j)x_{i0} \right]$$

$$= \left( x_{i0} + \frac{\Delta x_i}{2} - x_i \right)(x_{j0} + \frac{\Delta x_j}{2} - x_j)$$

$$\Rightarrow -\frac{\Delta x_i \Delta x_j}{4} \leq L_{A_{x_i x_j}}(x_i, x_j) - x_i x_j \leq \frac{\Delta x_i \Delta x_j}{4},$$

$$\Rightarrow |L_{A_{x_i x_j}}(x_i, x_j) - x_i x_j| \leq \frac{\Delta x_i \Delta x_j}{4}.$$ 

$\square$

**Theorem 2.4.** The linear function $L_{A_{x_i x_j}}(x_i, x_j)$ is the optimal linear approximation for $x_i x_j$, where $x_{i0} \leq x_i \leq x_{i0} + \Delta x_i, x_{j0} \leq x_j \leq x_{j0} + \Delta x_j$.

**Proof:**

Let $y = ax_i + bx_j + c$ be the optimal linear approximation of $x_i x_j$ within the subdomain $R = \{(x_i, x_j) \mid x_{i0} \leq x_i \leq x_{i0} + \Delta x_i \text{ and } x_{j0} \leq x_j \leq x_{j0} + \Delta x_j\}$. Let $E(x_i, x_j) =$
$x_i x_j - ax_i - bx_j - c$ be the linearization error function between $y$ and $x_i x_j$. Since $y$ is optimal,

$$\max_{(x_i, x_j) \in R} E(x_i, x_j) = \min_{(x_i, x_j) \in R} E(x_i, x_j).$$

Let

$$E_{\text{range}} = \max_{(x_i, x_j) \in R} E(x_i, x_j) - \min_{(x_i, x_j) \in R} E(x_i, x_j).$$

The maximum error for $y$ is $\frac{E_{\text{range}}}{2}$ within $R$.

$$E(x_i, x_j)$$

$$= x_i x_j - ax_i - bx_j - c$$

$$= (x_i - b)(x_j - a) - ab - c$$

The function $E(x_i, x_j)$ has the value $-ab - c$ at two lines $x_i = b$ and $x_j = a$. These two lines divide the x-y domain into 4 subdomains, $S_1, S_2, S_3$ and $S_4$. $E(x_i, x_j)$ is also divided into 4 parts, $e_1, e_2, e_3$ and $e_4$.

In $S_1$: $x_i > b, x_j > a$, $e_1$ monotonically increases as $x_i, x_j \to \infty$.

In $S_2$: $x_i < b, x_j > a$, $e_2$ is identical to the $e_1$ with a $180^\circ$ rotation along the $x_j$-axis.

In $S_3$: $x_i < b, x_j < a$, $e_3$ is identical to the $e_1$ with a $180^\circ$ rotation along the $y$-axis.

In $S_4$: $x_i > b, x_j < a$, $e_4$ is identical to the $e_1$ with a $180^\circ$ rotation along the $x_i$-axis.

The domain $R$ can possibly span 1, 2, or 4 subdomains by lines $x_i = b$ and $x_j = a$, as illustrated in Figure 2.6. Let $x_{i_0} + \Delta x_i - b = p \Delta x_i, x_{j_0} + \Delta x_j - a = q \Delta x_j$. Without lost of generality, we assume $p \geq \frac{1}{2}, q \geq \frac{1}{2}$, and the coverage of $R$ on 4 possible subdomains can be analyzed by the following 4 cases.

In case 1, $p > 1, q > 1$: $R$ spans only 1 subdomain. The linearization error is

$$\frac{E_{\text{range}}}{2}$$
Figure 2.6: Relation between the subdomain $R$ and two lines $x_i = b$ and $x_j = a$

$$
\begin{align*}
\mathcal{E}(x_i, x_j) & = \frac{1}{2} \left[ E(x_i, x_j) \big|_{x_i=x_{i0}+\Delta x_i, x_j=x_{j0}+\Delta x_j} - E(x_i, x_j) \big|_{x_i=x_{i0}, x_j=x_{j0}} \right] \\
& = \frac{1}{2} \left[ p\Delta x_i \cdot q\Delta x_j - ab - c - (p - 1)\Delta x_i \cdot (q - 1)\Delta x_j + ab + c \right] \\
& = \frac{1}{2} \left( p\Delta x_i \Delta x_j + q\Delta x_i \Delta x_j - \Delta x_i \Delta x_j \right) \\
& > \frac{\Delta x_i \Delta x_j}{2}.
\end{align*}
$$

In case 2, $p > 1$, $q < 1$: $R$ spans 2 subdomains. The linearization error is

$$
\begin{align*}
\mathcal{E} \mathcal{R} & = \frac{1}{2} \left[ E(x_i, x_j) \big|_{x_i=x_{i0}+\Delta x_i, x_j=x_{j0}+\Delta x_j} - E(x_i, x_j) \big|_{x_i=x_{i0}, x_j=x_{j0}} \right] \\
& = \frac{1}{2} \left[ p\Delta x_i \cdot q\Delta x_j - ab - c - (p - 1)\Delta x_i \cdot (q - 1)\Delta x_j + ab + c \right] \\
& = \frac{p\Delta x_i \Delta x_j}{2} \\
& > \frac{\Delta x_i \Delta x_j}{2}.
\end{align*}
$$

Case 3, $p < 1$, $q > 1$ is similar to case 2. The linearization error is also greater than $\frac{\Delta x_i \Delta x_j}{2}$. In case 4, $p < 1$, $q < 1$: $R$ spans 4 subdomains.

(1) If $E(x_i, x_j) \big|_{x_i=x_{i0}+\Delta x_i, x_j=x_{j0}} < E(x_i, x_j) \big|_{x_i=x_{i0}, x_j=x_{j0}+\Delta x_j}$ the linearization error is

$$
\begin{align*}
\mathcal{E} \mathcal{R} & = \frac{1}{2} \left[ E(x_i, x_j) \big|_{x_i=x_{i0}+\Delta x_i, x_j=x_{j0}+\Delta x_j} - E(x_i, x_j) \big|_{x_i=x_{i0}, x_j=x_{j0}} \right] \\
& = \frac{1}{2} \left[ p\Delta x_i \cdot q\Delta x_j - ab - c - (p - 1)\Delta x_i \cdot (q - 1)\Delta x_j + ab + c \right] \\
& = \frac{p\Delta x_i \Delta x_j}{2} \\
& > \frac{\Delta x_i \Delta x_j}{2}.
\end{align*}
$$
\[ \frac{1}{2}[E(x_i, x_j)|x_i=x_{i0}+\Delta x_i, x_j=x_{j0}+\Delta x_j - E(x_i, x_j)|x_i=x_{i0}+\Delta x_i, x_j=x_{j0}] \]
\[ = \frac{1}{2}[p\Delta x_i \cdot q\Delta x_j - ab - c - p\Delta x_i \cdot (q-1)\Delta x_j + ab + c] \]
\[ = \frac{p\Delta x_i \Delta x_j}{2}. \]
\[ \frac{\Delta x_i \Delta x_j}{4} \leq \frac{p\Delta x_i \Delta x_j}{2} < \frac{\Delta x_i \Delta x_j}{2}. \]

(2) Similarly, if \( E(x_i, x_j)|x_i=x_{i0}+\Delta x_i, x_j=x_{j0} > E(x_i, x_j)|x_i=x_{i0}, x_j=x_{j0}+\Delta x_j \) the linearization error is also greater than \( \frac{\Delta x_i \Delta x_j}{4} \) and less than \( \frac{\Delta x_i \Delta x_j}{2} \).

Since \( y = ax_i + bx_j + c \) is the optimal linear approximation, case 1, 2, and 3 which have greater error are impossible. The error of optimal linear approximation is as in case 4, \( \geq \frac{\Delta x_i \Delta x_j}{4} \).

\[ \square \]

With similar arguments, we have the following property for the linear approximation of \( ax_ix_j, LA_{ax_ix_j}(x_i, x_j) \).

**Corollary 2.2.** (1) \( LA_{ax_ix_j}(x_i, x_j) = a \cdot LA_{ax_ix_j}(x_i, x_j) \).

(2) The linear function \( a \cdot LA_{ax_ix_j}(x_i, x_j) \) is the optimal linear approximation for the function \( ax_ix_j \), where \( x_{i0} \leq x_i \leq x_{i0} + \Delta x_i \), and \( x_{j0} \leq x_j \leq x_{j0} + \Delta x_j \).

2.2.3. Linearizing a quadratic function

Based on the linearization techniques for the square and product terms, we present a simple method to obtain a linear approximation for a quadratic function,

\[ y = a + \sum_{i=1}^{n} b_i x_i + \sum_{i,j=1}^{n} c_{ij} x_i x_j. \]
We first compute the corresponding linear approximation as,

\[ LA_y = a + \sum_{i=1}^{n} b_i x_i + \sum_{i,j=1}^{n} LA_{c_{ij} x_i x_j}(x_i, x_j), \]

where \( LA_{c_{ij} x_i x_j}(x_i, x_j) \) is the linear approximation for \( c_{ij} x_i x_j \).

Since the corollary 2.1 and 2.2,

\[ \sum_{i,j=1}^{n} LA_{c_{ij} x_i x_j}(x_i, x_j) = \sum_{i,j=1}^{n} c_{ij} \cdot LA_{x_i x_j}(x_i, x_j), \]

hence we have a linear approximation for the quadratic function \( y \) as,

\[ LA_y = a + \sum_{i=1}^{n} b_i x_i + \sum_{i,j=1}^{n} c_{ij} \cdot LA_{x_i x_j}(x_i, x_j). \]

The linearization of \( y \) is analytically derived by simply summing up the unchanged constant and linear terms and the approximated linear functions of all quadratic terms. It keeps the constant and linear terms unchanged. Each quadratic term \( c_{ij} x_i x_j \) is approximated by Eq. 2.3 if \( i = j \) and by Eq. 2.5 if \( i \neq j \). However, the summation does not preserve the optimality of the linear approximation because the locations of the actual maximum error of each term may not be the same.

Since the linearization of each term can be analytically derived in constant time, the total time for linearizing a quadratic function is only \( O(n^2) \), where \( n \) is the number of variables in performance models. This is much more efficient than the linear least-square data fitting technique described in Section 2.1 for linearizing an arbitrary function.
2.3. Error Measurement and Designable Space Partitioning

In addition to improving the efficiency, another objective of our approach is to improve the accuracy on representing the acceptability region. In this section, we demonstrate how to partition the designable space so that our piecewise linear representation will globally satisfy the accuracy tolerance. In piecewise constant approaches [40], [41] [39], and [37], the designable space is partitioned uniformly by using sensitivity studies to determine the unit partition size on each designable's coordinate, so that the representation error of each performance model can meet its accuracy requirement. Since the sensitivity study is done for one designable at a time at a fixed location, the actual representation error may accumulate and be inaccurate [40] [41] [39]. Sam-Harould also suggests using sensitivity studies and iterative experiment for space partitioning [37]. Not only the similar problem could occur, but also a mechanism is needed to derive accuracy tolerance for intermediate variables to ensure accuracy of the representation.

The above difficulty results from the fact that the approximation error of the piecewise constant representation depends on not only the size of each partitioned subspace but also the location of the place it is measured. In Figure 2.7, there are two piecewise constant representation examples for a function, $y = x^2$, by two designable space partitionings. The left one is accurate when $x$ is small, but the inaccuracy increases as $x$ increases. In order to make the representation accurate throughout the whole space, $x_0 \leq x \leq x_0 + \Delta x$, the partitioning needs to be much finer like the one on the right. As a result, the complexity of the representation increases.
Figure 2.7: Error in piecewise constant representation

In the following, we present a worst case maximum error bound for our linearization technique. It leads us to an uniform space partitioning that guarantees the accuracy requirement is met. Let $L_A_y$ be the linear approximation function derived by the proposed linearization technique for a model function $y = a + \sum_{i=1}^{n} b_i x_i + \sum_{i,j=1}^{n} c_{ij} x_i x_j$, within a space subdivision $\{X | X_0 \leq X \leq X_0 + \Delta X\}$, where $X_0 \in D_1 \times D_2 \times \cdots \times D_n$, $D_i$ is the domain of the variable $x_i$, and $\Delta X = (\Delta x_1, \Delta x_2, \ldots, \Delta x_n)$. A worst case bound of the maximum error for $L_A_y$ is shown in the following lemma.

**Lemma 2.1.** For $X_0 \leq X \leq X_0 + \Delta X$, we have

$$|L_A_y - y| \leq \sum_{i=1}^{n} \frac{|c_{ii}|}{8} \Delta x_i^2 + \sum_{i \neq j; i,j=1}^{n} \frac{|c_{ij}|}{4} \Delta x_i \Delta x_j.$$ 

**Proof**

$$L_A_y - y = \sum_{i=1}^{n} c_{ii} \cdot (L_A x_i^2(x_i) - x_i^2) + \sum_{i \neq j; i,j=1}^{n} c_{ij} \cdot (L_A x_i x_j(x_i, x_j) - x_i x_j)$$

$$|L_A_y - y| \leq \sum_{i=1}^{n} \text{Max}(|c_{ii} \cdot (L_A x_i^2(x_i) - x_i^2)|)$$
Because of the above lemma, we have the uniform worst case bound property on the maximum error of the proposed linearization technique in the following theorem.

**Theorem 2.5.** If the designable space \( D_1 \times D_2 \times \cdots \times D_n \) is uniformly partitioned by \( \Delta X = (\Delta x_1, \Delta x_2, \ldots, \Delta x_n) \), and the worst case bound of the maximum linearization error from lemma 2.1 for any space subdivision is less than the maximum allowed error tolerance for the performance model \( y \), then the whole piecewise linear approximation function computed by the proposed linearization technique satisfies the accuracy requirement of the model \( y \).

**Proof**

From the lemma 2.1, we know the worst case maximum error bound is independent of the location of each space subdivision. As long as the size of each space subdivision is the same, the worst case maximum error bound is identical. Hence, if the linear approximation of any single subdivision satisfies the accuracy requirement, the whole piecewise linear function must also satisfy.

\( \square \)

Based on the Theorem 2.5, we have the key ideas of an algorithm for space partitioning: *partition uniformly* and *use the worst case maximum error bound of any*
space subdivision to satisfy the accuracy requirement. The procedure Find_worst-case_partitioning_box() is a greedy algorithm. It computes a maximum partitioning box to minimize the complexity of the space partitioning. The maximum partitioning box also needs to be small enough to make sure all models' linearization error will be less than accuracy tolerances. This procedure repeatedly partitions the space until the worst case maximum error bound is less than the error tolerance. To minimize the partitioning, one greedy method is to choose a dimension that can reduce the worst case maximum error bound as much as possible. The heuristic we implemented for line 6 of the procedure is to compute the worst case maximum error bound for each individual dimension and choose the dimension with the worst bound. For a reason that will be explained in chapter 3, we have the following restriction in finding a space partitioning for each model: the piecewise linear representation of each model should have the same designable space partitioning. In the procedure, the value $E_{max_i}^{worst}$ is the worst case maximum error bound for the performance model $y_i$ for the current unit partitioning box $\Delta X$. The value $T_i$ represents the maximum error tolerance allowed for a performance model $y_i$.

Procedure Find_worst-case_partitioning_box()

begin
1 Initialize the partition count of each dimension to 1;
2 Initialize $\Delta X$ as the whole input domain;
3 while there is at least one model whose $E_{max_i}^{worst} > T_i$
4 Find the model $j$ with the largest worst case max error v.s. error tolerance ratio, $E_{max_j}^{worst}/T_j$
5 Find the dimension $d$ that has the worst impact on worst case estimated error
for model $j$

Add one to partition count of dimension $d$, and update $\Delta X$.

end

2.3.1. An improvement to reduce the number of space subdivisions

While the procedure `Find_worst-case_partitioning_box()` will guarantee that all piecewise linear models meet accuracy requirements, using the worst case maximum error bound is really not necessary. It can over-partition the designable space, and increase the size of the piecewise linear models more than necessary. The best alternative is to use the actual maximum error in every space subdivisions to ensure accuracy. Finding the exact maximum error for linearization of quadratic is a quadratic programming problem. Solving a quadratic program for each space subdivision is very time consuming. In the following, we propose an improved method to efficiently use actual maximum error to partition the designable space.

For a designable space subdivision $s_1 : X_0 \leq X \leq X_0 + \Delta X$, let $LA_{x_i^2,X_0}(x_i)$ be the linear approximation for $x_i^2$ and $E_{x_i^2,X_0}(X) = LA_{x_i^2,X_0}(x_i) - x_i^2$ be the error function between $LA_{x_i^2,X_0}(x_i)$ and $x_i^2$. Similarly, for $x_ix_j$, we define the linear approximation $LA_{x_i,x_j,X_0}(x_i, x_j)$ and the corresponding error function $E_{x_i,x_j,X_0}(X) = LA_{x_i,x_j,X_0}(x_i, x_j) - x_ix_j$. We use $E_{y,X_0}(X) = LA_y - y$ to denote the error function between the model function $y$ and its linear approximation $LA(y)$. In the following lemma, we show that for another space subdivision $s_2 : X_0 + \delta X \leq X \leq X_0 + \delta X + \Delta X$, $s_1$'s error function is identical to $s_2$'s error function translated by $\delta X$. 


Lemma 2.2.

$$E_{y,x_0}(X) = E_{y,x_0+\delta X}(X + \delta X).$$

Proof

The error function for the model function $y$ within $X_0 \leq X \leq X_0 + \Delta X$ can be expanded as,

$$E_{y,x_0}(X) = \sum_{i=1}^{n} c_i \cdot E_{x_i^2,x_0}(X) + \sum_{i \neq j : i,j=1}^{n} c_{ij} \cdot E_{x_ix_j,x_0}(X). \quad (2.6)$$

(1) For the first half of the right hand side of the equation 2.6, we have

$$E_{x_ix_j,x_0}(X)$$

$$= LA_{x_i^2,x_0}(x_i) - x_i^2$$

$$= \frac{1}{8} \Delta x_i - (x_{i0} + \frac{\Delta x_i}{2})^2$$

$$= \frac{1}{8} \Delta x_i - (x_{i0} + \delta x_i + \frac{\Delta x_i}{2} - (x_i + \delta x_i))^2$$

$$= LA_{x_i^2,x_0+\delta X}(x_i + \delta x_i) - (x_i + \delta x_i)^2$$

$$= E_{x_i^2,x_0+\delta X}(X + \delta X).$$

(2) For the second half of the right hand side of the equation 2.6, we have

$$E_{x_ix_j,x_0}(X)$$

$$= LA_{x_ix_j,x_0}(x_i, x_j) - x_ix_j$$

$$= (x_{i0} + \frac{\Delta x_i}{2} - x_i)(x_{j0} + \frac{\Delta x_j}{2} - x_j)$$

$$= (x_{i0} + \delta x_i + \frac{\Delta x_i}{2} - (x_i - \delta x_i))(x_{j0} + \delta x_j + \frac{\Delta x_j}{2} - (x_j + \delta x_j))$$

$$= LA_{x_ix_j,x_0+\delta X}(x_i + \delta x_i, x_j + \delta x_j) - (x_i + \delta x_i)(x_j + \delta x_j)$$

$$= E_{x_ix_j,x_0+\delta X}(X + \delta X).$$
By (1) and (2), we prove $E_{y,x_0}(X) = E_{y,x_0+\delta X}(X + \delta X)$.

Based on the above lemma, we prove the following theorem. The actual maximum errors of different space subdivisions of the same size $\Delta X$ are the same. That is, the actual maximum error of a space subdivision is independent of its location.

**Theorem 2.6.**

$$
\max_{x_0 \leq x \leq x_0 + \Delta X} (|E_{y,x_0}(X)|) = \max_{x_0+\delta X \leq x \leq x_0+\delta X + \Delta X} (|E_{y,x_0+\delta X}(X)|).
$$

**Proof**

From the lemma 2.2

$$E_{y,x_0}(X) = E_{y,x_0+\delta X}(X + \delta X),$$

we have

$$
\max_{x_0 \leq x \leq x_0 + \Delta X} (|E_{y,x_0}(X)|) = \max_{x_0 \leq x \leq x_0 + \Delta X} (|E_{y,x_0+\delta X}(X + \delta X)|).
$$

Also since

$$
\max_{x_0 \leq x \leq x_0 + \Delta X} (|E_{y,x_0+\delta X}(X + \delta X)|) = \max_{x_0+\delta X \leq x \leq x_0+\delta X + \Delta X} (|E_{y,x_0+\delta X}(X)|),
$$

the theorem follows.

\qed

Because of the uniform maximum error property in the Theorem 2.6, if the designable space is partitioned by the size $\Delta X = (\Delta x_0, \Delta x_1, \ldots)$, and if there exist a space subdivision whose linearization error satisfies the required accuracy, then
the piecewise linear approximation over the whole designable space satisfies the required accuracy. We call the size $\Delta X$ box a \textit{standard partitioning box}. Based on this observation, our approach partitions the designable space with exactly the same size as the standard partitioning box. By controlling the error within this box, the user controls the accuracy of the whole piecewise linear representation. In other words, if the user later changes the error tolerance, our approach simply compute another standard partitioning box and designable space partitioning to satisfy the user's accuracy requirements. It is obvious that there exist many possible $\Delta X$. Since one of our objective is to minimize the complexity of each model's representation, we present a greedy algorithm in the procedure \textit{Find standard-partitioning box}. It first uses the procedure \textit{Find worst-case partitioning box} to find a partitioning which guarantees the piecewise linear models to be accurate. It then iteratively relaxes the partitioning until a standard partitioning box is found. Like the procedure \textit{Find worst-case partitioning box}, it computes the worst case maximum error bound for each individual dimension, and uses them to choose dimensions for relaxing the partitioning. Because it starts with a worst case partitioning, the curvature of each performance model is smooth within a space subdivision. We use the Monte Carlo method (random sampling) to estimate the real maximum errors $E_{max_i}$ for the linearization of performance models $y_i$, $i = 1, \ldots, m$. Uniform sampling is also an alternative. The value $T_i$ represents the maximum error tolerance allowed for performance model $y_i$.

\begin{verbatim}
Procedure \textit{Find standard partitioning box}()
begin

\end{verbatim}
Use `Find_worst-case_partitioning_box()` to find a partitioning \( P = (p_1, \ldots, p_n) \), where \( p_i \) is the number of partitions on dimension \( i, i = 1, \ldots, n \);

repeat

find a model \( j \) with the least \( \frac{T_j}{E_{max_j}} \) ratio;
find a dimension \( d \) such that \( p_d > 1 \) and has least impact

on the maximum error in model \( j \);

\( P' = (p_1, \ldots, p_d - 1, \ldots, p_n) \);
compute \( E_{max_k}, k = 1, \ldots, m \) based on \( P' \);
if \( E_{max_k} < T_k, \forall k = 1, \ldots, m \)

\( P = P' \);
else

return \( P \);
end

2.4. An Adaptive Linearization Algorithm

Once a standard partitioning box is found, the next step is to use its size to partition the designable space and find a piecewise linear representation for each model. A piecewise linear representation can be obtained by uniformly dividing the designable space with the standard partitioning box, and finding a linear approximation function within each subdivision. However, one can further reduce the size of the piecewise linear representation by considering performance specifications. In this section, we present an algorithm to compute a compact piecewise linear representation and store it in a binary space partitioning tree. The size of the tree depends on the size of the acceptability region.
Let $LA_{y_k}$ and $E_{max}^{\text{worst}}$ be the linear approximation function and the corresponding worst case maximum error bound for the quadratic model $y_k = \sum a_{kn} + \sum b_{kn}x_{kn} + \sum c_{kij}x_{ki}x_{kj}$ in a rectangular designable space subdivision $B = \{X | X_0 \leq X \leq X_0 + \Delta X\}$. Two values $U_k$ and $L_k$ are the maximum and minimum values for $LA_{y_k}$ within $B$. We denote $S_{k}^{\text{max}}$ and $S_{k}^{\text{min}}$ as the $y_k$'s maximum and minimum device performance specifications. The following theorem presents two key observations used in the algorithm.

**Theorem 2.7.** For all performance models $y_k, k = 1, \ldots, m$,

1. if $L_k - E_{max}^{\text{worst}} \geq S_{k}^{\text{min}}$ and $U_k + E_{max}^{\text{worst}} \leq S_{k}^{\text{max}}$, then the space subdivision $B$ is completely inside of the acceptability region;

2. if $L_k - E_{max}^{\text{worst}} \geq S_{k}^{\text{max}}$ or $U_k + E_{max}^{\text{worst}} \leq S_{k}^{\text{min}}$, then the space subdivision $B$ is completely outside of the acceptability region.

**Proof**

Let $R_{k}^{\text{max}}$ and $R_{k}^{\text{min}}$ be the maximum and minimum of the real performance output for $y_k$ within $B$

1. Since

\[ |R_{k}^{\text{min}} - L_k| \leq E_{max}^{\text{worst}}, \]

\[ \Rightarrow R_{k}^{\text{min}} - L_k \geq -E_{max}^{\text{worst}}, \]

\[ \Rightarrow R_{k}^{\text{min}} \geq L_k - E_{max}^{\text{worst}}. \]

If $L_k - E_{max}^{\text{worst}} \geq S_{k}^{\text{min}}$, then $R_{k}^{\text{min}} \geq S_{k}^{\text{min}}$. 

Similarly,

$$|R_k^{\text{max}} - U_k| \leq E_{\text{max}}^{\text{worst}},$$

$$\Rightarrow R_k^{\text{max}} \leq U_k + E_{\text{max}}^{\text{worst}}.$$  

If $U_k + E_{\text{max}}^{\text{worst}} \leq S_k^{\text{max}}$, then $R_k^{\text{max}} \leq S_k^{\text{max}}$. Since $S_k^{\text{max}} \geq R_k^{\text{max}} \geq R_k^{\text{min}} \geq S_k^{\text{min}}$, we know that all performance specifications are satisfied, that is, $B$ must be inside of the acceptability region.

(2) Since

$$|R_k^{\text{min}} - L_k| \leq E_{\text{max}}^{\text{worst}},$$

$$\Rightarrow R_k^{\text{min}} - L_k \geq -E_{\text{max}}^{\text{worst}},$$

$$\Rightarrow R_k^{\text{min}} \geq L_k - E_{\text{max}}^{\text{worst}}.$$  

If $L_k - E_{\text{max}}^{\text{worst}} \geq S_k^{\text{max}}$, then $R_k^{\text{min}} \geq S_k^{\text{max}}$. That is, $S_k^{\text{min}}$ is not satisfied within $B$. $B$ must be outside of the acceptability region.

Similarly,

$$|R_k^{\text{max}} - U_k| \leq E_{\text{max}}^{\text{worst}},$$

$$\Rightarrow R_k^{\text{max}} \leq U_k + E_{\text{max}}^{\text{worst}}.$$  

If $U_k + E_{\text{max}}^{\text{worst}} \leq S_k^{\text{min}}$, then $R_k^{\text{max}} \leq S_k^{\text{min}}$. That is, $S_k^{\text{max}}$ is not satisfied within $B$. $B$ must be outside of the acceptability region.

$\square$

From the above theorem, we know that the complete uniform designable space partitioning is not necessary. If a larger subdivision satisfies the condition in the above theorem, it is either completely inside or outside. Since our goal is to identify
the acceptability region, not an accurate piecewise linear model throughout the whole
designable space, we can accept some inaccuracy within the piecewise linear model
representation. As a result, our piecewise linear representation is adaptive, and its
complexity depends on the size of the acceptability region.

The algorithm is described by two procedures, Find_Piecewise_Linear_Rep() and
Helper(). The first procedure is the driver routine and the second one computes
the representation. The algorithm recursively divides the designable space in half
at a chosen dimension and finds linear approximations for all performance models
for both halves. If one of the conditions in the above theorem is true or the size of
the current subdivision is smaller than the size of the standard partitioning box in
all dimensions, a linear approximation function is recorded for each model for this
subdivision. Otherwise, a dimension is chosen for further partitioning. The same
heuristic used in procedure Find_Worst-Case_Partitioning_Box() to find a dimension
for further partitioning can also be used in line 10 of the procedure Helper().

Procedure: Find_Piecewise_Linear_Rep()

begin
    \( B = \text{Find.Standard.Partitioning.Box}() \);
    compute the maximum partition of each dimension of designable space;
    Helper(whole.designable.space);
end
Procedure: Helper(subdivision $D$)

begin

1. find $LA_{y_k}$ and $E_{\text{max}}^\text{worst}$ for all models $y_k$ in $D$;

2. compute $U_k$ and $L_k$;

3. case 1: if $L_k - E_{\text{max}}^\text{worst} \geq S_k^{\text{min}}$ and $U_k + E_{\text{max}}^\text{worst} \leq S_k^{\text{max}}$ for all $k$

   add $LA_{y_k}$ to $y_k$’s piecewise linear representation $\text{Tree}_{\text{bsp}}(y_k)$ for all $k$

   return;

4. case 2: if $L_k - E_{\text{max}}^\text{worst} \geq S_k^{\text{max}}$ or $U_k + E_{\text{max}}^\text{worst} \leq S_k^{\text{min}}$ for all $k$

   return;

5. case 3: otherwise

   if $D$’s size larger than $B$ in some dimensions;

   find the dimension $i$, that has the most impact on error;

   cut $D$ into $D_1$ and $D_2$ along the dimension $i$;

   update $\Delta X$;

   Helper($D_1$);

   Helper($D_2$);

6. otherwise

   add $LA_{y_k}$ to $y_k$’s piecewise linear representation, $\text{Tree}_{\text{bsp}}(y_k)$, for all $k$

   and return;

end

To make the algorithm more efficient, some heuristics can be applied. For example, one heuristic of choosing a dimension is to first choose a performance variable $y_k$, such that its linearization error to error tolerance ratio is the greatest, and then
choose a dimension that has the most impact on $y_k$'s linearization error. Also, while finding a dimension for further partitioning, only those dimension in which the current subdivision has a larger size than the standard partitioning box are considered.

Here, we conclude the presentation of the linearization of a quadratic performance model by pointing out the third main advantage of our representation: the capability of online re-linearization based on a different accuracy requirement. In the piecewise constant approach, different accuracy tolerance requires the users to figure out a different designable space partitioning by using sensitivity studies or other heuristics. But in our approach, since all our previously mentioned algorithms can be easily implemented, user can easily change the accuracy tolerance and get another piecewise linear representation for each device performance variable without sensitivity studies. This property gives users a great flexibility to experiment with different accuracy tolerance online.

2.5. Binary Space Partition Tree

We use a binary space partition (BSP) tree to represent a piecewise linearized model. The BSP tree is popular in many application areas; in particular, they are common in computer graphics. Its application includes hidden surface removal in the painter's algorithm [19], shadow generation [7], set operations on polyhedra [31] [44], and visibility preprocessing for interactive walkthroughs [43].

A BSP tree is often used to store objects in space by partitioning the space into subdivisions until there is exactly one object in each subdivision. A two-dimensional example is in Figure 2.8. This figure shows a binary space partition (BSP) for a
set of objects in the plane, together with the tree that corresponds to the BSP. The two-dimensional plane is recursively cut in half by a line: first we split the entire plane with \( l_1 \), then we split the half-plane on the left side of \( l_1 \) with \( l_2 \) and the half-plane on the right side of \( l_1 \) with \( l_3 \), and so on. Each internal node of its BSP tree corresponds to a splitting line. Each leaf node stores an object. The choice of BSP tree for our piecewise linear model is natural. Similarly, in our adaptive linearization algorithm, the space is also recursively cut into two subdivisions in a dimension. The only difference is that our splitting line is always orthogonal to the coordinate axis.

In each leaf node, we store a linear function for each space subdivision. An example of using BSP for the piecewise linear model is shown in Figure 2.9. The size of the tree is the total number of pieces in the piecewise linear model. The height of the tree is \( \sum_i \lfloor \log_2 s_i \rfloor \), where \( s_i \) is the number of partitions for the dimension \( x_i \).
Figure 2.9: An example BSP tree for piecewise linear representation
CHAPTER 3

ACCEPTABILITY REGION COMPUTATION

This chapter presents the computation, representation and visualization of acceptability regions. An acceptability region is computed from performance requirements and the piecewise linear performance models proposed in the chapter 2. Similar to other approaches [37] [40] [41] [39], our acceptability region consists of a set of smaller acceptability subregions. However, instead of restricting each subregion to a rectangular hyperbox, we choose a much more generalized shape, a convex polytope. With this new representation, the complexity of our acceptability region is greatly reduced. This representation is presented in Section 3.1. Section 3.2 describes an algorithm to compute the acceptability region. Section 3.3 discusses the concept of visualizing the acceptability region in the polytope based representation, and Section 3.4 presents polytope projection algorithms for two-dimensional visualization.

3.1. A New Polytope Representation for Acceptability Subregion

Previously in piecewise constant approaches [37] [40] [41] [39], the acceptability region is represented as a set of rectangular hyperboxes. Larger hyperboxes are used to fill in the inner portion of the acceptability region. Smaller ones are used to approximate the boundary of the acceptability region more accurately. Often because of accuracy requirements and the curvature of the acceptability region, a lot of small boxes are needed. The representation presented in this chapter also consists of a set of smaller acceptability subregions. Instead of many small hyperboxes, we use polytopes to represent the boundary of the acceptability region. Because the polytope is
more generalized in its shape than the rectangular hyperbox, it greatly reduces the complexity of the acceptability region. Each polytope is represented by a system of linear inequalities.

Deriving Acceptability Subregions

In the adaptive linearization process of Section 2.4, each piecewise linear performance model is constructed based on an identical designable space partitioning. In this section, we demonstrate how to derive an acceptability subregion within each designable space subdivision.

Because of the Theorem 2.7, the linearization algorithm in Section 2.4 is adaptive. It can determine whether a designable space subdivision is either completely inside or outside the acceptability region. Because of this property, we divide the designable space subdivisions into three types:

- Type-1 subdivision, which is completely inside of acceptability because of the condition 1 of the Theorem 2.7.

- Type-2 subdivision, which is completely outside of acceptability because of the condition 2 of the Theorem 2.7.

- Type-3 subdivision whose relation to acceptability region is yet to be determined.
Feasibility Subregions

For each type 3 subdivision, there is a linear approximation function \(LA_{y_i}\) for each performance variable \(y_i, i = 0, \ldots, m\). In \(O(n)\) time, where \(n\) is the dimension of \(LA_{y_i}\), we can compute the maximum and minimum values of \(LA_{y_i}, U_i\) and \(L_i\), for a type 3 subdivision. Let \(E_{\text{max}_i}\) be the estimated maximum linearization error for \(y_i\) and \(S_i^{\text{max}}\) and \(S_i^{\text{min}}\) be the maximum and minimum performance specification of \(y_i\).

Note that for equality performance specification, we have \(S_i^{\text{max}} = S_i^{\text{min}}\). Because of the linearization error, we relax all specifications by the estimated error to ensure that the actual acceptability region is included in our computed acceptability region. That is, \(S_i^{\text{max}}\) becomes \(S_i^{\text{max}} + E_{\text{max}_i}\), and \(S_i^{\text{min}}\) becomes \(S_i^{\text{min}} - E_{\text{max}_i}\). For each type 3 subdivision, there is a feasible subregion for each performance variable \(y_i\). All points in this feasible subregion satisfy the specification on \(y_i\). There are three possible cases of feasible subregion of \(y_i\):

- **case 1**: The whole designable space subdivision is feasible to \(y_i\)'s specification if 
  \[S_i^{\text{max}} + E_{\text{max}_i} \geq U_i \geq L_i \geq S_i^{\text{min}} - E_{\text{max}_i}.
  
- **case 2**: The whole designable space subdivision is not feasible to \(y_i\)'s specification if 
  \[S_i^{\text{max}} + E_{\text{max}_i} \leq L_i \text{ or } S_i^{\text{min}} - E_{\text{max}_i} \leq U_i.
  
- **case 3**: Only a part of the designable space subdivision is feasible to \(y_i\)'s specification if 
  \[L_i < S_i^{\text{min}} - E_{\text{max}_i} < U_i < S_i^{\text{max}} + E_{\text{max}_i} \text{ or } S_i^{\text{min}} - E_{\text{max}_i} < L_i < S_i^{\text{max}} + E_{\text{max}_i} < U_i \text{ or } L_i < S_i^{\text{min}} - E_{\text{max}_i} < S_i^{\text{max}} + E_{\text{max}_i} < U_i.
  
An example of all three cases of feasible subregions is in Figure 3.1. For case 1, the whole rectangular designable space subdivision is the feasible subregion for \(y_i\). For case 2, the feasible subregion is empty. For case 3, we derive the feasible subregion of
Figure 3.1: Three cases of feasible subregion for a type 3 subdivision

$y_i$ from the linear approximation function for this designable space subdivision and the specification on $y_i$. Without loss of generality, let’s assume $L_i < S_i^{\min} - E_{\max i} < S_i^{\max} + E_{\max i} < U_i$. The corresponding feasible subregion of $y_i$ for the designable subdivision $b_k^{low} \leq x_k \leq b_k^{up}$ is represented by the set of inequalities,

\[ b_k^{low} \leq x_k \leq b_k^{up}, k = 0, 1, \ldots, n. \]

\[ LA_{y_i} \geq S_i^{\min} - E_{\max i} \]

\[ LA_{y_i} \leq S_i^{\max} + E_{\max i} \]

Geometrically, it is a convex polytope. Because the acceptability subregion within each designable space subdivision is the intersection of the feasible subregions of all performance variables, we further divide the type 3 designable space subdivisions into four more types:
• Type-3.1 subdivision is completely inside of the acceptability region if all performance variables’ feasible subregions belong to case 1.

• Type-3.2 subdivision is completely outside of the acceptability region if at least one performance variable’s feasible subregion belongs to case 2.

• Type-3.3 subdivision is partly inside of the acceptability region if all performance variables’ feasible subregions belong to case 3 and their intersection is not null.

• Type-3.4 subdivision is completely outside of the acceptability region if all performance variables’ feasible subregions belong to case 3 and their intersection is null.

To determine whether the intersection of all performance variables' feasible subregions is null or not, the procedure Compute_Acceptability_Subregion() first derives a linear inequality system $Lin_{Ineq}$ to represent the intersection. It then runs a linear program on the intersection to determine its feasibility. If the result is not null, it outputs the $Lin_{Ineq}$ to be included in the acceptability region.

**Procedure: Compute_Acceptability_Subregion()**

```
begin
    initialize an inequality system $Lin_{Ineq}$ with the boundary of the current designable space subdivision, $b_k^{low} \leq x_k \leq b_k^{up}, k = 0, 1, \ldots, n$.

    foreach performance specification, $y_i \geq S_i^{min}$;

    if $L_i < S_i^{min} - E_{max} < U_i$

        add the inequality: $LA_{y_i} > S_i^{min} - E_{max}$, to $Lin_{Ineq}$;
```

foreach performance specification, \( y_i \leq S_{i}^{\text{max}} \);

if \( L_i < S_{i}^{\text{max}} + E_{\text{max}} < U_i \)

add the inequality: \( LA_{y_i} < S_{i}^{\text{min}} + E_{\text{max}} \), to \( \text{Lin}_\text{ineq} \);

if \( \text{Lin}_\text{ineq} = \text{Null} \)

return \( \text{Null} \);

else

creates a \( \text{BSP}_\text{TreeAR} \)'s node, \( \text{asr} \), for \( \text{Lin}_\text{ineq} \);

return \( \text{asr} \);

end

As a result, each acceptability subregion within a type 1 or type 3.1 designable space subdivision is the whole rectangular hyperbox of the subdivision. On the other hand, type 2, type 3.2, and type 3.4 space subdivisions will not generate any acceptability subregion. Each acceptability subregion within a type 3.3 subdivision is represented by a linear inequality system \( \text{Lin}_\text{ineq} \). Each \( \text{Lin}_\text{ineq} \) will have \( 2n \) inequalities from the boundary of the designable space subdivision and at most \( 2m \) inequalities from the performance specification, where \( n \) is the dimensionality of designable space and \( m \) the number of performance models. The linear program needed in \( \text{ComputeAcceptabilitySubregion()} \) can be solved in \( O((m+n)^2 + (m+n)^{1.5} n) L) \) time, where \( L \) is the number of bits in the input [45].

3.2. Acceptability Region in Binary Space Partitioning Tree

The acceptability region employs the binary space partitioning tree to organize the acceptability subregions because the acceptability subregions are also separated by the same designable space partitioning used for performance models. It has the identical
BSP tree as the performance models except for the leaf nodes. It stores a system of linear inequalities for each acceptability subregion, instead of a linear approximation. The algorithm used to derive the acceptability region is described in Compute_AR(). Because all performance models’ BSP trees have the identical structure, the algorithm simply does a simultaneous depth-first traversal on all models’ BSP trees. At each traversal step, it intersects the feasible subregions of all performance variables to derive an acceptability subregion.

**Procedure: Compute_AR()**

```plaintext
begin
initialize BSP_TreeAR;
for i = 1... total number of designable space subdivisions
    foreach BSP_TreeAR and BSP_Tree_{model}(Q_k), k = 0... total number of models
        get to the next leaf node in the depth-first order;
    switch (the current designable space subdivision d_i) :
        case type 1 or 3.1: append an acceptability subregion of d_i;
        case type 2 or 3.2: append Null;
        otherwise: asr = Compute_Acceptability_Subregion();
        append asr;
end
```

Like the piecewise linear functions for performance models, the computation of the acceptability region is also adaptive. The checking for the type 1, 2, 3.1, 3.2 subdivisions can be done in $O(n)$ time, where $n$ is the number of all designable variables. The complexity of this procedure is dominated by solving linear programs in
checking for the type 3.3 and 3.4 space subdivisions. Since type 3.3 and 3.4 space subdivisions only occur at the boundary of the acceptability region, the number of linear programs $N_{LP}$ needed depends on the size of the acceptability region. This makes our polytope-based representation especially efficient when the size of the acceptability region is much less than the size of overall designable space.

In the following, we compare the complexity of the piecewise constant method by Saxena et al. with our piecewise linear method. Let $C_{LP}$ be the time complexity needed for solving a linear program. The time complexity for computing our polytope-based acceptability region is $N_{LP} \times C_{LP}$. The time complexity of the piecewise constant method is $N_{S} \times C_{S}$, where $N_{S}$ is the total number of designable space partitions, and $C_{S}$ is the time needed to evaluate all performance models. The cost of $C_{LP}$ is more expensive than $C_{S}$. But both are solvable in polynomial time. On the other hand, $N_{LP}$ can easily be exponential time better than $N_{S}$, for example, when the piecewise constant method needs $q$ subdivisions in each of $n$ dimensions and the piecewise linear method needs $q/2$ subdivisions in each dimension.

The advantage of the piecewise linear method depends on its ability to reduce the total number of the space partitions. When the dimension of the problem is low and the accuracy requirement is loose, the $N_{S}/N_{LP}$ ratio is low. In this situation, our method may not have any efficiency advantage and may even take longer time. Also, piecewise constant method is very easy to implement. Generally, the problem in this case is much simpler, both methods are sufficient. However, as the dimension of the problem increases or the accuracy requirement becomes higher, the $N_{S}/N_{LP}$ ratio increases. Often, in this case, the piecewise constant method suffers from the combinatorial explosion
problem in space partitioning much earlier than the piecewise linear method. Consequently, solving the problem becomes impossible in both time or space considerations. Because the complexity of the space partitioning in piecewise linear method grows much slower with respect to dimension and accuracy requirements, our method have better advantage in this situation.

3.3. Acceptability Region Visualization

After the models are linearized, and the acceptability region is computed, the third step in our approach is to let the designer visualize the acceptability region. Its purpose is to give the designer information about the computed acceptability region so that the designer can adjust the performance specification or designable space to obtain a more refined acceptability region. The display of the acceptability region must be accurate and clear because the designer needs to identify the acceptable area within the design space and the coordinate of the boundary. Two-dimensional projection is the simplest method to achieve this requirement. In this section, we discuss the visualization of the two-dimensional shadows and slices of the whole acceptability region.

Shadow and Slice

The two-dimensional shadow of an acceptability region is the orthogonal projection of the whole acceptability region on any two user selected dimensions. It provides an overall view of the whole region. However, it is hard for the designer to imagine those areas in the acceptability region that project to the inner part of the shadow. For this situation, the visualization of a slice of an acceptability region is especially useful. Since the acceptability region is stored in a binary space partitioning tree
based on the designable space partitioning, we can choose any subset of acceptability subregions to display. In a designable space partitioning, each dimension $x_i$ is divided into segments, $s_{i,1}, s_{i,2}, \ldots, s_{i,t}$, for $i = 1, 2, \ldots, n$. A two-dimensional slice of an acceptability region is the set of acceptability subregions that are within a chosen segment from each dimension $i$, $i \in \{1, 2, \ldots, n\} \setminus \{p, q\}$. Its visualization is to orthogonally project all subregions in the slice on the $p$-$q$ plane. By sequentially visualizing acceptability region slices with only a third dimension $r$ changing from one segment $s_{r,j}$ to the next segment $s_{r,j+1}$, one can see how the slices change along the dimension $r$ and gain better understanding about the acceptability region. An example is illustrated in Figure 3.2. Similarly, one can further extend the concept of slice to $k$ dimensions by fixing segment for any $n - k$ dimensions. The designer can select any two of the remaining $k$ dimensions to orthogonally project for visualization. In fact, the shadow is just a special case of slice with $k = n$. In the following subsection, without loss of generality, we only consider the visualization of a shadow or equivalently an $n$-dimensional slice, and the visualization for lower dimensional slice is similar.

Acceptability Region Projection

The acceptability region is a set of smaller acceptability subregions. To obtain its two-dimensional projection, we need to compute the union of the projections of all acceptability subregions. Each acceptability subregion is convex because it is the intersection of half spaces. Its two-dimensional projection must also be convex. However, the two-dimensional projection of an acceptability region, the union of convex polygons may not be convex. This makes its orthogonal projection more difficult.
Figure 3.2: Shadow and slice of a three dimensional acceptability region
We distinguish each acceptability subregion by its location; either it is a part of the boundary, or it is completely inside of the acceptability region. The projection of a boundary subregion may or may not be a rectangle, but the projection of internal subregion must be a rectangle. One approach is to project every acceptability subregion in conjunction with some heuristics to avoid some redundant projections. Since the whole designable space is partitioned into rectangular subspaces, the projection plane of two selected variables is also partitioned into rectangular boxes. The projection of each acceptability subregion must and can only be inside one box on the projection plane. The algorithm groups those acceptability subregions that will be projected onto the same box and then computes the projection for each group at a time. One early-abort heuristic is used to speed up the projection process: the projection of many acceptability subregions can be skipped if the algorithm encounters an acceptability subregion whose projection will be exactly the same as the 2-dimensional projection box. In other words, its projection will overlap all other projected images from all remaining acceptability subregions in this group. Many other early abort heuristics can also be added in projecting each group. The procedure Project_AR() describes the algorithm. The algorithm for projecting each acceptability subregion is discussed in Section 3.4.

Procedure: Project_AR()

begin

1  foreach rectangular box, b, in the projection plane
2  foreach acceptability subregion, asr, that will be projected inside b,
3  if asr is a rectangular hyperbox
3.4. Projection of a Convex Polytope

The projection of each acceptability subregion is discussed in this section. Each acceptability subregion is a convex polytope and represented by a set of inequalities. Several researchers have studied this problem. One of the earliest works on projecting a high dimensional polytope $P$ onto a lower dimensional plane is by Fourier in 1827 [17]. This algorithm projects a polytope of dimension $d$ to another polytope in any dimension $k$, $1 < k < d$. Both input and output polytopes are represented by sets of inequalities. This algorithm iteratively eliminates one variable in the inequality system at a time by the Gaussian elimination. But since we are dealing with inequalities instead of equalities, the multiplicative coefficients used during the Gaussian elimination must all be positive. The time complexity of Fourier's method is $O(f^{2d-k})$, where $f$ is the number of inequalities of $P$. For two-dimensional visualization, we still need to compute the set of vertices of the projected polygon, which is plagued by the fact that there are many redundant inequalities in the output of the Fourier's method.
Upper Bound

Amenta and Ziegler studied the worst case complexity of this problem and proved the following theorem [2].

**Theorem 3.1.** The $k$-dimensional projection of a convex polytope $P$ can be computed in $O(f^{d/2})$ time, for $1 < k < d$.

They also give a worst case optimal algorithm as follows.

**Algorithm:** Worst-Case Optimal()

begin
1  Compute $P$
2  foreach face $q$ of $P$
3       if $q$ is supported by a $k$-vertical hyperplane
4          add $q$ to $F$
5  return $F$
end

A hyperplane of the form, $a_1x_1 + a_2x_2 + \ldots + a_kx_k + b = 0$, is called $k$-vertical. A face $q$ is supported by a $k$-vertical hyperplane $H$ if $q$ is contained in $H$ and $P$ is contained in one of the halfspaces bounded by $H$. The time complexity of this algorithm is dominated by line 1, which can be done by Chazelle's convex hull algorithm [6].

An Output-Sensitive $k$-Dimensional Projection Algorithm

Although the worst case complexity takes exponential time, the size of the projection output is rarely exponential. Often, the size of the output decreases with $k$. In this
section, we study an algorithm which can take advantage of this situation: its complexity depends on the output size.

Ponce, et al, propose one of the best output-sensitive algorithms [32]. The algorithm, described in the procedure Contour_Tracing(), is divided into two phases. It first solves a linear program to find an initial vertex on the projected $k$-dimensional polytope in line 1. Any of the $k$ coordinates can be chosen as an objective function. From each vertex, the algorithm tracks the adjacent edges and records the vertical supporting hyperplanes. For each edge contained in a $k$-vertical supporting hyperplane, it finds the other endpoint for future visits.

**Algorithm: Contour_Tracing()**

begin

1. find a vertex $v$ that will also be a vertex of the projected polytope
2. insert $v$ to $S$
3. while $S \neq \emptyset$
4. take a vertex $v'$ from $S$ and put it in $D_v$
5. foreach edge $e$ adjacent to $v'$
6. if $e$ is contained by a $k$-vertical supporting hyperplane $h$
7. $D_h = D_h \cap h$
8. find the other endpoint of $e$, $v''$
9. if $v'' \notin D_v$
10. insert $v''$ to $S$

end

When the dimension of the problem $d$ is taken as a constant, line 1 can be done
in $O(f)$ time by Megiddo’s algorithm in [28]. For line 6, there are at most \( \binom{d-1}{d-k+1} \) supporting hyperplanes, the overall testing time takes constant time. Line 8 is done by intersecting each hyperplane of $P$ with the ray containing $e$. The point closest to $v'$ is the other endpoint $v''$. This requires time complexity $O(f)$. Since the while loop in line 3 is executed exactly $t$ times, the overall time complexity is $O(tf)$.

However, for our application, $d$ is a variable and $k$ is 2. Megiddo’s algorithm is no longer a good choice for line 1. Its complexity is exponential in $d$. We can use Vaidya’s algorithm [45] instead. The complexity from line 3 to 10 is bounded by $O(t(f + d)d^3)$.

3.4.1. A New Algorithm for The Two-Dimensional Case

In this section, we propose a new algorithm to project a $d$-dimensional polytope $P$ onto two-dimensional $i$-$j$ plane. Because the two-dimensional projection of $P$, $P_{ij}$, is a convex polygon, one can use a sequence of polygon vertices in either clockwise or counterclockwise order to represent $P_{ij}$. The projection problem in two-dimensional case becomes finding the ordering of the sequential vertices.

The key idea of this algorithm is based on the following observation and explained in Figure 3.3. Let the vertices $v_1$ and $v_2$ be vertices of the projected polygon $P_{ij}$. The equation $\overline{v_1v_2} : ax_1 + bx_2 + c = 0$ is the line that contains $v_1$ and $v_2$. If there is a vertex $v_3$ of $P_{ij}$ between $v_1$ and $v_2$, we can set up a linear program with $P$ and use $ax_1 + bx_2 + c$ as an objective function. Solving this linear program finds the vertex $v_3$. The two-dimensional projection algorithm is illustrated in an example in Figure 3.4. In the first picture, we use two linear programs to find the minimum and maximum
Figure 3.3: Using linear programming to find polygon vertices, $v_1, v_4$ on the $x_i$ coordinate. The algorithm sequentially identifies the polygon vertices in the counterclockwise order. It first searches for the vertices from vertex $v_1$ toward vertex $v_4$, then searches for the vertices from vertex $v_4$ toward vertex $v_1$. In picture 2, we use the linear function for $v_1v_4$ as an objective function and compute a linear program to search toward the right hand side. A new vertex $v_3$ is identified.

In picture 3, we continue using the same technique, but use $v_1v_3$'s function as the objective function. Another new vertex $v_2$ is found. In picture 4, we don't find any new vertex between $v_1, v_2$, between $v_2, v_3$, and between $v_3, v_4$. We sequentially record $v_1, v_2, v_3$ as the first part of the sequence of the polygon vertices in a stack. In picture 5, we use the function for $v_4v_1$ as an objective function and compute a linear program to search toward the left, and we identify a new vertex $v_5$. In picture 6, we can't find any vertex between $v_4, v_5$ and between $v_5, v_1$. We record $v_4$ and $v_5$ in a stack. Because the starting vertex $v_1$ is reached, the algorithm stops.

The algorithm is described in **Polygon Expanding**. The step 2 makes the algorithm first search for new vertices from $v_{minij}$ to $v_{maxij}$, then search for new vertices
Figure 3.4: An example for the new two-dimensional projection algorithm from $v_{max_{ij}}$ back to $v_{min_{ij}}$. In each while loop, the algorithm tries to find a new vertex $v_{found}$ so that $v_1$, $v_{found}$, and $v_2$ on the polygon $P_{ij}$ will be in counterclockwise order. The cross product in step 5 is to ensure the vertex $v_3$ is on the right-hand side of the vector $v_1v_2$. The point $v_3$ is to guide the linear program to search for a new vertex $v_{found}$ in the right-hand-side direction of the vector $v_1v_2$ as described from the line 7 to the line 10. As a result, the sequence of the polygon vertices computed by this algorithm is in counterclockwise order.

Algorithm: Polygon_Expanding()

begin

1 find the min and max vertices, $v_{min}$, $v_{max}$, on $x_i$ coordinate;
2 push vertices $v_{\min ij}, v_{\max ij}$, then $v_{\min ij}$ into $S$
3 while the size of $S \geq 2$
4 pop $v_1 = (x_{i1}, x_{j1})$ and $v_2 = (x_{i2}, x_{j2})$ from $S$;
5 find a vertex $v_3$, such that $x_1 x_3 \times x_1 x_2 > 0$;
6 find $L(x_i, x_j) = \bar{v}_1 \bar{v}_2$;
7 if $L(v_3) > 0$
8 use $L$ as the maximize criterion and find a vertex $v_f$ in $P$.
9 else
10 use $L$ as the minimize criterion and find a vertex $v_f$ in $P$.
11 if $v_{fi} = v_1$ or $v_2$
12 push $v_1$ in $V$, push $v_2$ back to $S$;
13 else push $v_2, v_{fi}$, then $v_1$ back to $S$.
14 return $V$;
end

The vertex $v_3$ of line 5 can be found in constant time. The running time of this algorithm is dominated by $2t$ linear program computations. We can again use Vaidya's algorithm. Its time complexity is $O((f + d)d^2 + (f + d)^{1.5}d)L)$, where $L = \log_2$ (largest absolute value of the determinant of any square sub-matrix of $A$) + $\log_2$ (max $c_i$) + $\log_2$ (max $b_i$) + $\log_2$ (f + d), $P = \{x : AX \geq b\}$, and $c$ is the coefficient of the objective function. One main advantage of our algorithm is its simplicity. Complex algorithms are not needed to solve numerical problems except for a linear program solver, many solvers are widely available in both academy and industry.
CHAPTER 4

DESIGN OPTIMIZATION WITH ACCEPTABILITY REGIONS

An optimization method commonly followed is to use a nonlinear optimizer to iteratively search for an optimal device design point [16] [3] [42] [35]. However, it is well known that the computation of the global optimal solution is very difficult and not guaranteed. Many algorithms only converge to a local optimal solution, and this is often not satisfactory. Using multiple starting points may improve the quality of the solution, but often requires expensive computations. Also because it is not clear whether the failure is due to a solution not existing or due to the optimization algorithm not converging. A failure is very difficult to interpret in the nonlinear optimization.

In this chapter, we first discuss an adaptive design method based on our polytope-based acceptability region for design optimization, and then discuss its advantages in detail.

A flow diagram of the adaptive optimization design paradigm based on our acceptability region representation is shown in Figure 4.1. Unlike the optimizer-based paradigm, it computes the set of all feasible solutions. It encourages the designer to adaptively study the problem, examine the acceptability region, and derive a globally optimal solution.
4.1. Global Optimization with Acceptability Regions

One of the most important advantages of computing the acceptability region is the ability to compute a globally optimal solution. The algorithm and its complexity depend on the acceptability region’s representation. In the previous representation, as well as the polytope-based representation, the acceptability region can be regarded as a set of subregions. A globally optimal solution can be computed by examining every piece of acceptability subregions. Since the total number of subregions is finite, a globally optimal solution is guaranteed to be found.

We first consider the effects of the piecewise constant representation on the acceptability region based optimization. For the grid-based representation by Saxena, et. al, [40] [41] [39], one simple algorithm is the exhaustive search. The algorithm evaluates the objective function at the center of all unit hyperboxes within the region and returns the point with the best value. One advantage of this algorithm is its simplicity: it does not require pre-processing the objective function. On the other hand, its complexity depends on the size of its acceptability region representation. Similarly, for the Sam-Harould’s representation [37], the simple exhaustive search scheme
is also applicable. However, due to the additional temporary variables introduced in the ternary function transformation, the complexity of the search space increases considerably.

In our polytope-based representation, an acceptability region is a set of convex polytopes. One algorithm for finding a globally optimal solution is to first apply the piecewise linearization technique on the objective function if it is quadratic, we can then use a linear program to search for an optimal point in each polytope. The complexity of this algorithm depends on the total number of linear programs needed. On the other hand, the complexity of the piecewise constant representation depends on the number of objective function evaluations. The relative efficiency of the two approaches depends on their numbers of partitions of the design space.

4.2. Extracting Design Information from Acceptability Regions

Visualization

The computation of the acceptability region leads itself to a very powerful visualization of the solution space by showing the space of all possible solutions in many ways.

1. The designer can examine the acceptable design space of any two variables, \( x_i \) and \( x_j \), within the complete acceptability region. This design flexibility is presented by showing the shadow of the acceptability region on dimensions \( x_i \) and \( x_j \).

2. The designer can study the design flexibility of any two variables while some of other variables are fixed to a value or an interval. This is shown by projecting
a slice of the acceptability region on any two dimensions.

3. The designer can examine a series of acceptability region slices with one or more variables' values being varied to study how the acceptability region changes according to these variables.

Approximate Information on Device Performance

One other type of information is the approximate ranges of all device performances. Because we have a piecewise linear approximation function for each device performance, we can obtain the minimum and maximum values of each device performance variable from the linear approximation of each performance equation in \( O(n) \) time within each designable space subdivision. Because each approximation function for each performance parameter within a subdivision is linear, the value of performance variable linearly increases from the minimum location to the maximum location. With this performance trend information, the designer can learn how the response of a device performance variable changes across the acceptability region. This is especially useful if presented with the projection of a slice of acceptability region. An example is illustrated in Figure 4.2. The arrow in each designable space subdivision represents the direction from the location of the minimum to the maximum.

Approximate Information on Objective Function

With the piecewise linearization technique, we can linearize a quadratic objective function across the acceptability region and also provide the similar approximate trend information. The capability to study the acceptability region before the optimization process is another advantage of this design paradigm. Better understanding of the distribution of different design goals within the acceptability region can help the
Figure 4.2: Showing the performance approximation along with the acceptability region

designer formulate a better composite objective function to discover better design points.

4.3. Design with Acceptability Regions

1. After examining the performance, objective function, and acceptability region information from the acceptability region, the designer can go back to step 1 to either re-specify the performance requirement or modify the designable domain of interest.

- This allows the designer to adaptively study the relation among the performance models, specifications, and the set of all solutions. In addition, providing "all solutions" gives the designer much more information about the design flexibility than providing just one solution as in the optimizer-based methods. This is especially important for solving a design problem where formally formulated constraints are typically only a subset of all constraints.
• This also permits the designer to refine the acceptability region toward the more optimal region even before the optimization step.

2. Discovering optimal design alternatives is one of the most important functionalities for solving design problems. By combining the acceptability region examination and optimization, the designer can conduct optimizations in different acceptability regions with different trade-offs on device performance specifications.

3. This paradigm also allows fully automatic optimization without any interference. That is, after the acceptability region is computed, an objective can be supplied to find a globally optimal solution without examining or refining acceptability regions.

4.4. Consistent Acceptability Regions After Editing

An acceptability region contains the set of all solutions that meet the performance requirement. However, not all points within the region are equally desirable. For example, in semiconductor device design, some design points might have better parametric yield, which results in fewer wafers lost. For some un-formulated constraint information like this, the designer would need to directly edit the acceptability region to focus on the more desirable region.

Sam-Harould's representation represents the acceptability region as a set of partial solution regions. Each partial region contains value assignments for three variables and each assignment can be extended to a full solution without backtracking. This property is maintained by a global consistency checking algorithm. However, if one
of the partial solution region is modified, all other partial solution regions need to be reconfigured to ensure the global consistency. This could become an inconvenient overhead for online acceptability region editing. On the other hand, both ours and Saxena's representations represent the acceptability region as a whole in full dimensions. The acceptability region maintains its global consistency after modifications.

4.5. Handling More Difficult and High Dimensional Problems

Often in the early stage of the design, without any knowledge of the acceptability region, the designer uses the full designable space, desired performance specifications and desired linearization error tolerances to formulate a device decomposition problem. But the problem can be computationally infeasible because of the extra high dimensionality or the complex designable space partitioning. In this section, we illustrate how to use our design paradigm to handle this situation. The key idea is to keep the complexity of finding the acceptability region computationally feasible during each iteration by manipulating many complexity factors. We identify the following four complexity factors.

- the dimensionality of the designable space.

- the performance model's linearization error tolerance.

- the size of the designable space.

- the ranges of performance specifications.

Of course, one can reduce the dimensionality of the designable space by setting some designable variables to a fixed value. But the resulting acceptability region is a subset of the real acceptability region. For this reason, we only consider the other
three factors.

One advantage of our linearization technique is that the accuracy tolerance is adjustable online. Once a problem is found infeasible, the designer can immediately loosen the accuracy requirement to simplify the problem. From this, the designer can obtain a less accurate but complete acceptability region. After examining the acceptability region, the designer can focus on the region of more interest by tightening up the designable space and specifications. In the meantime, since the complexity of the problem is reduced, the degree of accuracy can be increased. This adaptive scheme allows the designer to tackle the originally infeasible problem through managing its complexity online. On the other hand, adjusting the accuracy tolerance online is inconvenient and inaccurate for those approaches that require sensitivity studies to partition the space.
CHAPTER 5

IMPLEMENTATION AND EXPERIMENTAL RESULTS

This chapter demonstrates the feasibility of proposed algorithms by applying their implementations to two practical examples.

In Section 5.1, we describe an implementation of the linearization algorithm presented in Section 2.1. It linearizes any model without any function type assumption. We also apply it to Sam-Harould's civil engineering example and compare with their results. Section 5.2 implements our adaptive linearization method for quadratic models. To help verify the correctness of our result, we implemented a simplified visualization tool to visualize the resulting acceptability region in Section 5.3.

5.1. General Function

The implementation of the linearization method in this section is for any models without any function type assumption. It is a feasibility study to test out the general linearization idea. Its overall mechanism is identical to the linearization method for quadratic model. It first partitions the designable space and finds the piecewise linear models. The acceptability region is then computed from specifications and piecewise linear models. However, the algorithm differs in three parts.

- It recursively partitions each space subdivision until its maximum linearization error is less than accuracy tolerance.

- It implements a simple nonlinear optimization algorithm, down-hill simplex, to estimate error [33].
• It implements a linear least-square data fitting algorithm, called singular-value decomposition, to compute linear approximation models [33] [25].

In computing acceptability region, we need to solve linear programs to check whether each subregion is null. We use a package called cdd+ [20] to estimate the running time. We tested this implementation with a practical example from civil engineering [37]. The model is as follows.

\[
\begin{align*}
  u &< (3.18e-05 * hs + 0.0054) * s \\
  hs &> 137.70 - 8633e - 05 * s + 5.511e - 05 * s^2 - 8.358e - 07 * s^3 \\
  p &= u + 9.62e - 05 * (0.0417 * w)^{1.5161} \\
  hb &> 0.077 * (p * w^2)^{0.3976} \\
  hb &> 0.0168 * (p * w^3)^{0.2839}
\end{align*}
\]

The running time comparison is shown in Table 5.1. In this example, our implementation provided a very encouraging result for the polytope representation and linearization technique.

<table>
<thead>
<tr>
<th>Sam-Harould</th>
<th>Piecewise Linearization Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000 sec</td>
<td>20 sec</td>
</tr>
<tr>
<td>on SG Iris 4D</td>
<td>on Pentium</td>
</tr>
<tr>
<td>[37]</td>
<td>133MHz, 32 Mbyte</td>
</tr>
</tbody>
</table>
5.2. Quadratic Function

In this section, we implement the algorithms described in chapter 2 and 3. The implementation is experimented on an example from Texas Instruments. There are four models $p_1, p_2, p_3, p_4$. The ranges of the input variables for these models are:

- $x_0: [2, 5]$
- $x_1: [0.2, 0.4]$
- $x_2: [4.5, 7.5]$
- $x_3: [5, 7.5]$
- $x_4: [0.35, 0.7]$
- $x_5: [4.5, 7]$
- $x_6: [60, 90]$
- $x_7: [0.3, 0.8]$
- $x_8: [45, 65]$
- $x_9: [0.08, 0.12]$
- $x_{10}: [19, 19.7]$
- $x_{11}: [6, 16]$
- $x_{12}: [2.0, 3.0]$
- $x_{13}: [0.01, 0.033]$
- $x_{14}: [18.4, 19.3]$
- $x_{15}: [-5, 5]$
- $x_{16}: [19.4, 20.2]$
- $x_{17}: [11, 16]$
- $x_{18}: [2.22, 2.33]$
- $x_{19}: [0.03, 0.07]$
\[ x_{20} : [60, 200] \]

\[ p_{1} : 1.786026e+00 + x_{6} * 1.806890e - 02 + x_{7} * -1.056673e + 01 + x_{9} * -1.094333e + 01 + x_{13} * 6.834174e + 01 + x_{7} * x_{7} * -1.113158e + 01 + x_{0} * x_{0} * 8.237464e - 03 + x_{1} * x_{1} * -5.155017e - 01 + x_{2} * x_{2} * -9.914060e - 04 + x_{3} * x_{3} * 6.863032e - 03 + x_{4} * x_{4} * -1.526533e - 01 + x_{10} * x_{10} * -4.399522e - 04 + x_{12} * x_{12} * -1.778376e - 01 + x_{16} * x_{16} * -5.263994e - 04 + x_{6} * x_{7} * 2.127185e - 02 + x_{6} * x_{9} * 1.484841e - 02 + x_{6} * x_{0} * 7.664875e - 04 + x_{6} * x_{1} * -3.311773e - 03 + x_{6} * x_{8} * -5.060910e - 05 + x_{6} * x_{3} * 5.129954e - 04 + x_{6} * x_{4} * -6.845729e - 03 + x_{6} * x_{10} * -8.232116e - 04 + x_{6} * x_{11} * -3.761619e - 05 + x_{6} * x_{13} * -8.058324e - 02 + x_{7} * x_{9} * -1.002115e + 01 + x_{7} * x_{0} * -8.413689e - 02 + x_{7} * x_{3} * -7.644092e - 02 + x_{7} * x_{4} * 4.555725e - 01 + x_{7} * x_{10} * 6.965932e - 01 + x_{7} * x_{11} * 2.690372e - 02 + x_{7} * x_{12} * -9.037509e - 01 + x_{7} * x_{16} * 2.850212e - 01 + x_{7} * x_{17} * 1.889663e - 02 + x_{7} * x_{13} * 6.543499e + 01 + x_{9} * x_{4} * 6.802525e - 01 + x_{9} * x_{5} * 1.430434e - 01 + x_{9} * x_{10} * -4.203836e - 01 + x_{9} * x_{11} * -2.034775e - 02 + x_{9} * x_{12} * 9.307410e - 01 + x_{9} * x_{16} * 9.339996e - 01 + x_{9} * x_{17} * 9.808753e - 02 + x_{9} * x_{13} * -3.404069e + 01 + x_{0} * x_{8} * -5.534526e - 04 + x_{0} * x_{3} * -3.181371e - 03 + x_{0} * x_{4} * 1.589207e - 02 + x_{1} * x_{8} * 3.848045e - 03 + x_{1} * x_{3} * 2.145171e - 02 + x_{8} * x_{4} * 4.170114e - 03 + x_{2} * x_{4} * 1.020436e - 02 + x_{3} * x_{4} * -8.935669e - 02 + x_{3} * x_{13} * 2.489470e - 01 + x_{4} * x_{5} * 1.375523e - 02 + x_{4} * x_{13} * -2.204435e + 00 + x_{5} * x_{10} * -1.330865e - 03 + x_{5} * x_{15} * -1.020080e - 04 + x_{10} * x_{11} * -5.061928e - 04 + x_{10} * x_{12} * 5.141146e - 02 + x_{10} * x_{13} * -4.906592e + 00 + x_{11} * x_{12} * 5.275359e - 03 + x_{11} * x_{13} * -1.824785e - 01 + x_{12} * x_{13} * 3.861749e + 00; \]

\[ p_{2} : 3.565567e + 02 + x_{6} * -1.320852e + 01 + x_{7} * 3.753634e + 03 + x_{3} * -4.637417e + ]

\[ 01 + x_{11} \times 1.411978e + 01 + x_{13} \times -2.627376e + 04 + x_{6} \times x_{6} \times 1.489285e - 01 + x_{7} \times x_{7} \times 3.600812e + 03 + x_{0} \times x_{0} \times -5.779429e + 00 + x_{1} \times x_{1} \times 2.755583e + 02 + x_{8} \times x_{8} \times 2.397668e - 02 + x_{2} \times x_{2} \times 2.185416e - 01 + x_{4} \times x_{4} \times 1.886713e + 02 + x_{5} \times x_{5} \times 1.353375e + 00 + x_{10} \times x_{10} \times 1.274262e + 01 + x_{16} \times x_{16} \times 1.147829e + 01 + x_{6} \times x_{7} \times 1.415607e + 01 + x_{6} \times x_{9} \times 9.140468e + 00 + x_{6} \times x_{4} \times -5.423588e - 01 + x_{6} \times x_{5} \times -1.262065e - 01 + x_{6} \times x_{10} \times -7.551021e - 01 + x_{6} \times x_{11} \times -1.909253e - 02 + x_{6} \times x_{12} \times 1.010936e + 00 + x_{6} \times x_{16} \times -2.676435e - 01 + x_{6} \times x_{20} \times 5.959880e - 03 + x_{6} \times x_{13} \times -3.854386e + 01 + x_{7} \times x_{9} \times 3.911853e + 03 + x_{7} \times x_{0} \times 2.684759e + 01 + x_{7} \times x_{8} \times -1.180916e + 00 + x_{7} \times x_{3} \times 2.624537e + 00 + x_{7} \times x_{4} \times -1.339231e + 02 + x_{7} \times x_{5} \times -1.977415e + 01 + x_{7} \times x_{10} \times -3.422984e + 02 + x_{7} \times x_{11} \times -1.078664e + 01 + x_{7} \times x_{12} \times 3.607257e + 02 + x_{7} \times x_{16} \times -1.041938e + 02 + x_{7} \times x_{17} \times -7.108149e + 00 + x_{7} \times x_{20} \times 6.854731e - 01 + x_{7} \times x_{13} \times -2.714087e + 04 + x_{9} \times x_{10} \times 6.374815e + 02 + x_{9} \times x_{11} \times 2.193534e + 01 + x_{9} \times x_{12} \times -7.140824e + 02 + x_{9} \times x_{16} \times -6.736906e + 02 + x_{9} \times x_{17} \times -4.370057e + 01 + x_{9} \times x_{13} \times 2.772373e + 04 + x_{0} \times x_{8} \times 1.504986e - 01 + x_{0} \times x_{3} \times 1.095116e + 00 + x_{0} \times x_{20} \times 1.436606e - 02 + x_{1} \times x_{8} \times -1.416652e + 00 + x_{1} \times x_{4} \times -4.184527e + 01 + x_{8} \times x_{4} \times -1.899732e + 00 + x_{3} \times x_{4} \times 3.736519e + 01 + x_{3} \times x_{5} \times 6.026198e - 01 + x_{3} \times x_{11} \times -1.001206e - 01 + x_{3} \times x_{20} \times 8.226341e - 03 + x_{4} \times x_{5} \times -5.832738e + 00 + x_{4} \times x_{20} \times -7.627495e - 02 + x_{4} \times x_{13} \times 1.061087e + 03 + x_{10} \times x_{11} \times 3.237912e - 01 + x_{10} \times x_{12} \times -2.073914e + 01 + x_{10} \times x_{16} \times -1.806871e + 01 + x_{10} \times x_{17} \times -7.472632e - 01 + x_{10} \times x_{20} \times -3.963127e - 02 + x_{10} \times x_{13} \times 2.987605e + 03 + x_{11} \times x_{12} \times -1.933644e + 00 + x_{11} \times x_{16} \times -6.608477e - 01 + x_{11} \times x_{17} \times -3.683585e - 02 + x_{11} \times x_{13} \times 9.485668e + 01 + x_{12} \times x_{16} \times 1.634095e + 01 + x_{12} \times x_{13} \times -2.464076e + 03 + x_{16} \times x_{17} \times 1.129463e + 00 + x_{16} \times x_{13} \times -6.967644e + 02 + x_{20} \times x_{13} \times -2.290441e + 00; \]

\[ p_{3} : -3.311592e + 01 + x_{7} \times 1.529190e + 02 + x_{13} \times -9.598126e + 02 + x_{7} \times x_{7} \times \]
1.938514e+02+x1*x1*1.028750e+01+x2*x2*1.515092e−02+x3*x3*−1.172494e−
01+x5*x5*4.713154e−03+x10*x10*9.387996e−02+x11*x11*3.874359e−03+
12*x12*2.516527e+00+x16*16*7.156545e−02+x17*x17*9.490560e−03+x6*
x7*−1.741724e−01+x6*x3*−3.930909e−03+x6*x4*3.029393e−02+x6*x13* 4.228888e−01+x7*x9*1.445263e+02+x7*x10*2.811611e+00+x7*x1*−5.247915e+
00+x7*x8*7.989580e−02+x7*x3*1.724401e+00+x7*x4*−2.036262e+01+x7*
x10*−1.145088e+01+x7*x11*−4.296030e−01+x7*x12*1.589396e+01+x7*16*−4.577423e+
00+x7*x17*−2.760057e−01+x7*x13*−1.065325e+03+x9*x0* 2.525422e+00+x9*x3*1.170700e+00+x9*x4*−1.761145e+01+x9*x10*7.897324e+
00+x9*x11*2.031632e−01+x9*x16*−1.044624e+01+x9*x17*−1.474322e+00+
x0*x10*−7.822401e−02+x0*x11*−5.847297e−03+x0*x13*−8.214030e−02+
x0*x3*−3.365322e−01+x0*x4*−1.617668e−02+x2*x4*−2.116999e−01+x1*
x3*−3.365322e−01+x1*x4*−1.617668e−02+x2*x4*−2.116999e−01+x3*
x4*−3.790460e−01+x3*x11*−3.550755e−03+x3*x13*−5.838339e−03+x4*
x10*4.786718e−01+x4*x11*3.210009e−02+x4*x13*7.623436e+01+x10*
x11*1.114749e−02+x10*x12*−8.268017e−01+x10*x13*7.709264e+01+x11*
x12*−7.394247e−01+x11*x13*2.729277e+01+x12*x13*−7.005497e+01;

p4 : −1.463485e+02+x7*2.079079e+01+x9*−3.278581e+01+x10*1.455344e+
01+x11*9.745576e−02+x13*−4.402241e+02+x9*x9*−1.748849e+02+x0*
x0*1.423717e−02+x1*x1*−1.455764e+00+x8*x8*−9.895025e−05+x2*x2*−2.392524e−03+x3*
x3*8.923069e−03+x4*x4*−1.877309e+00+x10*x10*−3.306923e−01+x17*
x17*3.774192e−03+x14*x14*−5.480703e−03+x19*
x19*1.078979e+02+x20*x20−3.543722e−06+x6*x7*−4.732245e−02+x6*
The experimental result is shown in Table 5.2. The linearization error allowed for our linearization method ranges from 1 to 2% of the model output ranges. We compare it with what previously achieved. It is clear that an 8-dimensional problem was already hard. Each additional dimension means the problem will require 10
to 100 times as much running time and memory space. Solving a 20-dimensional problem is impossible. On the other hand, through our linearization technique, the acceptability region can be efficiently computed and represented in our polytope-based representation within minutes on a Pentium-based PC. Note that the performances of both machines are comparable. The performance benchmark result SPECint95 for Sparc 20 model 71 is about 2.5, and about 4.0 for Pentium 133.

Table 5.2: An experiment result on a quadratic function case

<table>
<thead>
<tr>
<th>TI's</th>
<th>Piecewise Linearization Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 dimensions</td>
<td>20 dimensions</td>
</tr>
<tr>
<td>in 1 hour</td>
<td>in 2 minutes</td>
</tr>
<tr>
<td>on Sun Sparc 20</td>
<td>on Pentium 133 MHz</td>
</tr>
</tbody>
</table>

5.3. Visualization

The objective of this visualization implementation is to provide the two-dimensional projection of the acceptability region to verify the correctness of the computed acceptability region. Because of that, we did not implement our polytope projection algorithm described in Section 3.4.1. Instead, we implemented a simpler approximation algorithm. For each two-dimensional projection of a polytope on the $x_i$-$x_j$ plane, we simply use a linear programming package to compute the minimum and maximum of $x_j$ values on a few fixed $x_i$ values to approximate the contour of the projected polygon. The output of the acceptability region projection consists of a set of polygons. A polygon drawing tool is written in Tcl/Tk [47][23]. It reads in a sequence of vertices for each polygon from an output file and displays them. The
following are a few pictures of acceptability region projections.

Performance specification 1:

\[ p_1 : [0.41, 0.42] \]
\[ p_2 : [301, 302] \]
\[ p_3 : [-8, -7.9] \]
\[ p_4 : [-1.1, -0.9] \]

Figure 5.1: An acceptability region for specification 1 on \( x_7 \)-\( x_6 \) plane

Performance specification 2:

\[ p_1 : [-1, 0] \]
\[ p_2 : [0, 300] \]
\[ p_3 : [-20, 20] \]
\[ p_4 : [-10, 5] \]
Figure 5.2: An acceptability region for specification 1 on $x_7-x_{17}$ plane

Figure 5.3: An acceptability region for specification 2 on $x_7-x_6$ plane
5.4. Summary

This chapter demonstrated the feasibility and efficiency of the piecewise linearization approach for the device decomposition problem and the constraint satisfaction problem. The implementations were applied to practical examples and compared with previous results.
CHAPTER 6

CONCLUSIONS

This chapter concludes this research with major contributions and future research directions of our methodology for the device decomposition problem and the constraint satisfaction problem.

6.1. Major Results

The major results are summarized as follows.

1. We proposed the concept of the piecewise linearization and demonstrated a novel adaptive linearization technique to approximate quadratic models with piecewise linear functions.

   - It is much more efficient than using the piecewise constant function for space partitioning because the linear function results in a better fit for the curvature of the quadratic performance model.

   - Unlike the previous methods which require sensitivity studies for the space partitioning, our method is completely automatic and can generate the piecewise linear functions online for any specified accuracy requirement.

   - Because the maximum errors of each piece of linear approximation are identical, the linearization technique can globally control the accuracy of the whole piecewise linear approximation.

2. We presented a polytope-based representation for acceptability regions.
• It can greatly reduce the complexity of representing an acceptability region. This property makes it especially useful when the dimensionality of the problem is high so that the existing methods are not applicable because of the time and space required.

• Its accuracy also provides a major improvement over the existing methods. It inherits the accuracy from the piecewise linear approximation models so that all acceptability subregions satisfy the accuracy requirements.

3. We demonstrated a method to visualize high-dimensional acceptability regions and reviewed the problem of projecting high-dimensional convex polytopes to a lower-dimensional space. A simple output sensitive algorithm for two-dimensional projection was proposed.

4. We examined the effects of the linearization technique and the polytope-based representation on the design optimization. It has following three types of advantages;

• Our method provides an adaptive design paradigm to iteratively examine the acceptability region and compute optimal design points.

• The polytope-based representation provides flexible combinations of different information to examine acceptability regions.

• The efficiency of our method allows the computation of globally optimal solutions in higher-dimensional cases, with the condition that the objective function is either linear or quadratic.

5. The experimental results show not only our method can provide time complexity speed-up but also enable us to process a 20-dimensional example while the
previous method was limited to 8-dimensional cases.

6.2. Future Research Extensions

In the following, we propose two future directions for further research along with our method.

1. There are situations in which the model is more complex than quadratic. More research in extending the analytic linearization method to either a higher degree of polynomial or other more general types of functions is needed to enlarge the applicability of the linearization method. Also, the objective function of the optimization may not be quadratic. The generalization can also be applied to the objective function to make the optimization process more flexible.

2. When the size of the resulting acceptability region becomes larger and more complex, the time required to project it can be unbearable to the user. A faster acceptability region projection method to avoid projecting redundant subregions becomes crucial to user friendliness.

3. The visualization of acceptability regions has been limited to two-dimensional projections in this dissertation. Because of the multidimensional nature of the acceptability region, displaying acceptability regions in more dimensions will be very useful for designers to obtain better understanding about the design space and help search for a better design point.

6.3. Summary

The device decomposition problem for semiconductor device design and the constraint satisfaction problem are very important and practical. Yet, they can be arbitrar-
ily difficult. This dissertation represents an attempt to provide a new and efficient approach. Specifically, it provides a piecewise linearization method for models, a polytope-based representation for acceptability regions, and algorithms for acceptability region visualization and device design optimization. The implementation has shown the feasibility of the presented approach for both types of problems.
REFERENCES


