HEURISTIC AND EXACT TECHNIQUES FOR SOLVING A TEMPERATURE ESTIMATION MODEL

by

Dale Lawrence Henderson

A Dissertation Submitted to the Faculty of the DEPARTMENT OF SYSTEMS AND INDUSTRIAL ENGINEERING
In Partial Fulfillment of the Requirements For the Degree of DOCTOR OF PHILOSOPHY
In the Graduate College
THE UNIVERSITY OF ARIZONA

2005
THE UNIVERSITY OF ARIZONA  
GRADUATE COLLEGE  

As members of the Dissertation Committee, we certify that we have read the dissertation 
prepared by Dale Lawrence Henderson 
entitled Heuristic and Exact Techniques for Solving a Temperature Estimation Model 
and recommend that it be accepted as fulfilling the dissertation requirement for the 
Degree of Doctor of Philosophy. 

_____________________________  Date: 7 Oct 2005  
Dr. J. Cole Smith  

_____________________________  Date: 7 Oct 2005  
Dr. Ronald Askin  

_____________________________  Date: 7 Oct 2005  
Dr. Jeffrey Goldberg  

_____________________________  Date: 7 Oct 2005  
Dr. Robert Indik  

_____________________________  Date: 7 Oct 2005  
Dr. Pitu Mirchandani  

Final approval and acceptance of this dissertation is contingent upon the candidate’s submission of the final copies of the dissertation to the Graduate College.  

I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement. 

_____________________________  Date: 7 Oct 2005  
Dissertation Director: Dr. J. Cole Smith
STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: DALE LAWRENCE HENDERSON
Acknowledgements

The author wishes to acknowledge the patience and support of Adrianna and Katherine Henderson. They have unselfishly endured much in support of the preparation of this document. In addition, David and Edward Henderson gave freely of their expert knowledge of programming and software development and Daniel Henderson contributed from his wealth of expertise in document preparation. This work would not have been possible without their help. Finally, many thanks are due Lawrence and Donna Henderson, who nurtured a child’s passion for finding out about things.
DEDICATION

This work is dedicated to the 2338 members of the United States military who, as of this date, have given their lives bringing the hope of freedom from tyranny to the people of Iraq and the hope of a just peace to the people of the world so that others might enjoy free and peaceful lives dedicated to scholarship.
# Table of Contents

List of Tables .................................................. 8

List of Figures .................................................. 9

Abstract .......................................................... 10

Chapter 1. INTRODUCTION ........................................ 11
  1.1. Literature Review ........................................... 12
      1.1.1. CTMs ................................................ 12
      1.1.2. Relevant nonlinear optimization techniques .......... 15
  1.2. Organization of the Dissertation ............................ 17
  1.3. Terminology and Symbols ................................... 17

Chapter 2. COMPACT THERMAL MODELS ............................ 22
  2.1. Electronic Chip Package .................................... 22
  2.2. Graph Representation ...................................... 23
  2.3. Thermal Behavior .......................................... 26
  2.4. Thermal Balance in a CTM .................................. 27
  2.5. Varying the Boundary Conditions ........................... 28
  2.6. Objective of the CTM ....................................... 29
  2.7. The CTM Optimization Model ................................. 31
  2.8. The CTM as a General Quadratic Programming Problem .... 31
  2.9. CTM Data .................................................. 32

Chapter 3. HEURISTIC TECHNIQUES ............................... 33
  3.1. Some Observations on the CTM Problem ...................... 33
  3.2. Perturbing a Single Value of $r_{ij}$ ....................... 34
  3.3. Effect of a Small Perturbation on the Objective Function 36
  3.4. Optimizing the Perturbed Objective Function ............... 37
      3.4.1. Analysis of $h(\delta)$ ............................. 37
      3.4.2. Inner optimization algorithm ....................... 42
      3.4.3. Overall heuristic algorithm .......................... 44
      3.4.4. Notes on the heuristic .............................. 45

Chapter 4. AN RLT APPROACH ................................... 47
  4.1. RLT ....................................................... 47
  4.2. Three Relaxations of CTM .................................. 50
Table of Contents—Continued

4.2.1. A convex quadratic programming relaxation of the quadratic CTM ........................................ 51
4.2.2. A linear programming relaxation of the quadratic CTM ......................................................... 53
4.2.3. A linear programming relaxation of the least absolute deviation CTM ................................... 54
4.3. The RLT algorithm ....................................................................................................................... 56

Chapter 5. IMPROVING THE PERFORMANCE OF RLT ................................................................. 61
5.1. Bounding the $T$-variables based on a problem upper bound ....................................................... 63
5.2. A reformulation using nonnegative variables .................................................................................. 64
5.3. A reformulation using fewer quadratic terms .................................................................................. 68
5.4. Valid inequalities that tighten the RLT relaxation .......................................................................... 69

Chapter 6. SOFTWARE DESIGN ......................................................................................................... 72
6.1. Software Framework ...................................................................................................................... 72
6.2. Functional Framework .................................................................................................................. 74
6.3. Software Components ................................................................................................................... 77
   6.3.1. The RltNode component ......................................................................................................... 77
   6.3.2. The CTMHeuristic component ............................................................................................... 79
   6.3.3. The RLTMaster component .................................................................................................. 81
   6.3.4. The RLTSolver component ................................................................................................... 82
   6.3.5. The DataReader component ................................................................................................. 83
   6.3.6. Extending the use of the software .......................................................................................... 84

Chapter 7. COMPUTATIONAL EXPERIENCE ..................................................................................... 85
7.1. Heuristic Results ............................................................................................................................ 86
   7.1.1. Computational time versus solution quality ............................................................................. 86
   7.1.2. Examining starting position ................................................................................................... 88
   7.1.3. Random restart ...................................................................................................................... 89
7.2. RLT Results .................................................................................................................................. 91
7.3. An experiment on stability ............................................................................................................ 92

Chapter 8. CONCLUSIONS & RECOMMENDATIONS ........................................................................ 94
8.1. Conclusions .................................................................................................................................... 94
8.2. Recommendations ........................................................................................................................ 94
   8.2.1. Decomposition techniques .................................................................................................... 95
   8.2.2. Improving lower bounds ....................................................................................................... 97

REFERENCES ......................................................................................................................................... 98
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1.1</td>
<td>Parameter definitions.</td>
<td>18</td>
</tr>
<tr>
<td>Table 1.2</td>
<td>Variable definitions.</td>
<td>19</td>
</tr>
<tr>
<td>Table 1.3</td>
<td>Computation parameter definitions.</td>
<td>20</td>
</tr>
<tr>
<td>Table 7.1</td>
<td>CTM data sets.</td>
<td>85</td>
</tr>
<tr>
<td>Table 7.2</td>
<td>Conditions for comparison.</td>
<td>87</td>
</tr>
<tr>
<td>Table 7.3</td>
<td>Comparison of CPU times and objectives under different conditions.</td>
<td>87</td>
</tr>
<tr>
<td>Table 7.4</td>
<td>Objective values for different starting positions.</td>
<td>89</td>
</tr>
<tr>
<td>Table 7.5</td>
<td>Progress of random restart over 50 trials.</td>
<td>90</td>
</tr>
<tr>
<td>Table 7.6</td>
<td>Results of an experiment using RLT.</td>
<td>92</td>
</tr>
<tr>
<td>Table 7.7</td>
<td>Results using randomly perturbed input data.</td>
<td>93</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 2.1. A two node CTM model. ........................................ 24
Figure 2.2. A star topology CTM model. ................................. 24
Figure 2.3. A partially shunted CTM model. ......................... 25
Figure 2.4. A clique topology CTM model. ............................. 26
Figure 2.5. Thermal diagram of a CTM. ................................. 27
Figure 3.1. Plot of $g(\delta)$ when $1/\sigma < \lambda/\mu$. ............... 38
Figure 3.2. Plot of $g(\delta)$ when $1/\sigma > \lambda/\mu$. ................ 39
Figure 3.3. Plot of $g(\delta)$ when case 3 holds. ..................... 40
Figure 3.4. Plot of $g(\delta)$ when case 4 holds. ..................... 40
Figure 3.5. Example of calculating region lower bounds. ........... 41
Figure 4.1. Three convex relaxations of the CTM. ................... 51
Figure 4.2. Flowchart of the RLT procedure. .......................... 59
Figure 5.1. Schematic description of the problem of lower bounds. . 61
Figure 5.2. A set of cuts based on the objective upper bound. ...... 63
Figure 5.3. An illustration of the effect of the $\tau$ formulation. .... 70
Figure 6.1. Software framework. ......................................... 73
Figure 6.2. Master class diagram. ....................................... 75
Figure 6.3. Node list and node structure. ............................... 76
Figure 6.4. The RLT node object. ..................................... 78
Figure 6.5. The CTM heuristic object. ................................. 80
Figure 6.6. The RLT master problem object. ......................... 81
Figure 6.7. Node list and node structure. .............................. 83
ABSTRACT

This dissertation provides several techniques for solving a class of nonconvex optimization problems that arise in the thermal analysis of electronic chip packages. The topic is of interest because in systems containing delicate electronic components both performance and reliability are impacted by thermal behavior. A modeling paradigm, called Compact Thermal Modeling (CTM), has been demonstrated to show promise for accurately estimating steady state thermal behavior without resorting to computationally intensive finite element models or expensive direct experimentation. The CTM is a network model that gives rise to a nonconvex optimization problem. A solution to this nonconvex optimization problem provides a reasonably accurate characterization of the steady state temperature profile the chip will attain under arbitrary boundary conditions, which allows the system designer to model the application of a wide range of thermal design strategies with useful accuracy at reasonable computational cost. This thesis explores several approaches to solving the optimization problem. We present a heuristic technique that is an adaptation of the classical coordinate search method that has been adapted to run efficiently by exploiting the algebraic structure of the problem. Further, the heuristic is able to avoid stalling in poor local optima by using a partitioning scheme that follows from an examination of special structure in the problem’s feasible region. We next present several exact approaches using a globally optimal method based on the Reformulation Linearization Technique (RLT). This approach generates and then solves convex relaxations of the original problem, tightening the approximations within a branch and bound framework. We then explore several approaches to improving the performance of the RLT technique by introducing variable substitutions and valid inequalities, which tighten the convex relaxations. Computational results, conclusions, and recommendations for further research are also provided.
Chapter 1

INTRODUCTION

This paper provides several approaches to solving a class of nonconvex optimization problems that arise in the thermal analysis of electronic chip packages. The thermal behavior of electronic chip packages is of concern to engineers developing electronic devices for several reasons. First, the reliability of the electronic components that make up a device is influenced by cycling between extreme thermal conditions. This cycling weakens solder joints whose failure is a significant causative mechanism in component malfunctions [74]. Second, high temperatures can induce instability and degraded performance in integrated circuits. Thus, an engineer tasked with designing an electronic device at the system or board level is interested in predictions of the steady state thermal characteristics of the electronic chip packages that are the principle heat generating components of the device. Specifically, the employment of temperature moderating solutions like heat sinks, air or fluid flow over components, and stand-off between packages and cases depends on predictions of the steady state temperatures that specific physical regions of the chip package or board will achieve. Given a reasonably accurate picture of the steady state thermal behavior of an electronic chip package, the designer can employ appropriate solutions to ensure that the device will operate within given thermal parameters. Ideally these predictive models should be robust and standardized to permit their extension to a wide variety of boundary condition designs, and to allow their inline use in an overall design process.

Several options are available for fulfilling the need for an accurate model for the steady state prediction of thermal behavior. One approach is to perform direct experimentation on a set of reference package designs under a set of initial conditions. This approach requires the fabrication of prototypes as well as carefully controlled
experimentation and measurement designed to characterize the package under a specific set of anticipated conditions. A second approach is to develop reasonably scaled finite element (FE) models to explore a particular combination of package geometry and material composition under a range of boundary heat dissipation conditions. The technique of concern in this dissertation involves the use of a representative rather than finite element model of the chip package. These models, called Compact Thermal Models (CTMs), represent the package as a network in which a set of boundary nodes serve as surrogates for physical regions of the chip, a junction node represents the chip’s power junction, and arcs permit thermal communication between adjacent nodes. CTMs are desirable because they allow accurate predictions of thermal behavior without the computational cost of finite element models or the expense of direct experimentation. Recent research into the use of CTMs is the topic of the first portion of the literature review provided in the next section.

1.1 Literature Review

This literature review is provided in two sections. The first concerns the development and current state of research into CTMs. The second provides an overview of nonlinear programming techniques, especially those that employ convex relaxations. This section also provides a review of work on the generalized quadratic programming problem, of which the CTM is a special case.

1.1.1 CTMs

The CTM modeling paradigm had its origin in attempts to enhance the reliability of electronic components that contained soldered, surface mounted, integrated circuits. By the early 1980’s, electronic devices had begun to make wide use of this manufacturing technique, and the heat generated by electronic chip packages was recognized as a primary factor impacting the reliability of these electronic systems. The problem
of modeling the thermal behavior of these systems was recognized as difficult, and several Finite Element (FE) techniques were proposed [74]. The focus of many of the early studies was on the effect of thermal cycling on the soldered junction pin or lead of the chip [19]. As early as 1985, there were attempts to bypass the complexity of FE models in favor of simplified aggregating models [33]. The 1988 paper by Anders [11], included a description of an electronic chip package as a set of regions related by a thermal resistance whose units were degrees Celsius per Watt, \([C/W]\). In the 1997 paper by Bar-Cohen and Krueger [16], the authors described the junction to air thermal resistance in terms of an empirical formula of the form \(q = (T_j - T_i)/R_{ij}\). That is, that the energy flow, \(q\), between two regions, \(i\) and \(j\), is equal to the difference in temperature between the regions, divided by the value of some physical thermal resistance parameter. This is a key modeling assumption in the CTM described in Chapter 2. Since energy is conserved, a network model of energy flow in the chip, given a set of thermal resistance values, could provide the nodal temperatures with very little calculation.

A fundamental shift in thermal modeling methods, due to Lasance, Vinke, and Parry was developed in [44] and [45]. In this work, the authors proposed a network description of the electronic chip package with thermal relationships between nodes described by this simple physical formula using the notion of a thermal resistance to explain heat flow through the network. In addition, they proposed that chip packages be characterized by that single set of resistances that best explained a predicted or reference temperature profile over a set of extreme boundary conditions. A concise description of the CTM model was presented in [71]. The DELPHI project, whose aim was to set a standard for the thermal modeling of electronic chip packages, grew out of this original notion of the network description, and by 1997 had produced a framework for thermal modeling [48, 52]. The progress of the DELPHI project was reported in two parts, [32, 43]. In conjunction with the reports on the DELPHI project, Lasance [38] made the argument for the adoption of the CTM as a modeling
paradigm. Lansance and Vinke [70] provided an overview of the advances made in the DELPHI work. Lansance [41] also provided a discussion of the accuracy with which CTMs might be expected to match experimental results. A similar project, called PROFIT [40], examined the non steady state behavior of packages. Aranyosi et al. [14] proposed a technique for the design of the boundary conditions under which to test a CTM.

Various topologies for the CTM network have been proposed and explored. Wang, Liou, and Sun [75] examined a two-node case in which only a junction node and an aggregate of the entire surface of the chip were represented. As an extension, Adams et al. [3] examined a “star” topology in which the junction node was connected to each of several boundary nodes, but in which the boundary nodes were not allowed to communicate thermally with one another. Shidore, Adams, and Lee [62] proposed a model that allowed the use of “shunts” between connected boundary nodes. The application of dedicated optimization software to the CTM problem was proposed by Lasance in [39]. Ortega and DeVoe [22, 23] developed an optimization methodology called the University of Arizona Compact Thermal Model (UACTM) and performed experiments that demonstrated that the CTM can provide results that are within 5% of those generated by a FE model. They extended this work using more sophisticated computational fluid dynamic models and provided confirmation of the fidelity of CTM solutions with Berhe in [24]. Further experimental comparisons were provided by Eveloy et al. [25]. Gabel and Ortega examined the transient behavior of electronic chip packages in [29]. A recent review of the state of CTM modeling was provided by Lasance in [42]. Smith et al. [63] proposed the heuristic technique described in Chapter 2 of this dissertation.


1.1.2 Relevant nonlinear optimization techniques

The CTM is a special case of the general nonconvex quadratic programming problem (QPP). Consequently it falls into a category of problems that have been thoroughly studied and in which there remains considerable interest. The problem is closely related to bilinear and biconvex programming problems and 0-1 quadratic programming problems. A thorough treatment of the general QPP was provided by Floudas and Visweswaran in [28]. Adjiman and Floudas [4] examined convex underestimators for twice differentiable problems. This dissertation is primarily concerned with deterministic approaches to solving the CTM; however, there exists a large body of research into the application of stochastic methods to solving general nonconvex problems of this type. The reader is referred to the work of Aarts [1, 2], on local search and on simulated annealing. The text [20] provides a detailed description of genetic algorithms.

We review a portion of the rich body of literature pertaining to the deterministic solution of problems of this type. The general nonconvex quadratic programming problem was shown to be NP-hard by Vavasis [69]. Pardalos and Vavasis [47] provided some further insight into the complexity of nonconvex quadratic programming. In the intervening years, several approaches to solving these problems have been proposed. The general idea of most of these approaches is to solve a sequence of convex relaxations of the original problem. One approach is to use semidefinite approximations to the indefinite problem. Kim and Kojima [34, 35] proposed an exact technique based on second order cone programming relaxations. Kojima and Levent [37] demonstrated that these methods are finitely convergent.

Van Voorhis and Al-Khayyal [68] described an approach that considers a transformation of the general problem into a problem that contains functions that are the difference of convex quadratic functions. They made a direct comparison of this technique to the RLT approach explored in this dissertation. More recently,
Van Voorhis [67] proposed a global optimization scheme using Lagrangian underestimates. Al-Khayyal [5, 6] provided several discussions of the linear complementarity problem and its relationship to the bilinear programming problem. Al-Khayyal [7] also proposed solving the linear complementarity problem as a bilinear program. Al-Khayyal, Larsen, and Van Voorhis [8] described a relaxation method for the nonconvex quadratic programming problem. Anjos [12, 13] discussed semidefinite relaxations of the satisfiability problem whose importance to the study of computational complexity was covered in [30]. Vandenbussche and Nemhauser [65, 66] provided a branch and cut algorithm for solving the nonconvex quadratic programming problem with (0,1) box constraints. A more general branch and cut approach was provided by Audet et al. [15]. Raber [51] studied the nonconvex all quadratic optimization problem and developed a simplicial branch and bound technique for solving the general quadratic programming problem in [50]. Visweswaran and Floudas [26, 27, 72, 73] provided a primal-relaxed dual approach to solving general quadratic programming problems called the Global Optimization Algorithm (GOP).

Between 1990 and 1992, an important new approach to solving nonconvex polynomial problems of arbitrary degree was proposed in work done by Sherali, Adams, Alameddine, and Tuncbilek. Sherali and Adams [54] explored the convex relaxations of 0-1 problems. Sherali and Alameddine [56] characterized the convex envelope of bivariate bilinear functions. In [59] Sherali and Tuncbilek proposed the RLT as a global optimization technique. This procedure was thoroughly examined in [55]. This work was extended to include the case of rational exponents in [53]. Sherali and Fraticelli [58] proposed a technique for enhancing RLT performance with semidefinite cuts. The Reformulation Linearization Technique for generating convex approximations is the subject of Chapter 4 of this paper, and is the technique we have chosen to generate exact solutions to the CTM problem.

The RLT approach has been applied to a number of problems. Sherali and Almeddine [57] applied the procedure to the bilinear programming problem, and Sherali and
Tuncbilek [60] to the general quadratic programming problem, and to invariate and multivariate polynomial problems [61]. Recently Amaral, Judice, and Sherali [10] applied the RLT procedure to the correction of an inconsistent system of linear constraints, and Amaral and Barahona [9] examined the connection between solving a least squares problem and correcting an inconsistent linear system. Lasserre [46] provided a comparison between semidefinite relaxations and LP relaxations as in RLT.

1.2 Organization of the Dissertation

The remainder of this dissertation is organized into chapters as follows: in the final section of this chapter we enumerate the terms, symbols, and definitions used throughout the paper to provide a convenient and consistent reference. In Chapter 2 we then present the formulation of the CTM model and make note of its relationship to similar problems. In Chapter 3 we demonstrate a heuristic technique for solving the CTM. In Chapter 4 we examine several approaches to using RLT to find an exact solution to the CTM problem. We explore the use of several techniques to enhance the performance of RLT in Chapter 5 by developing a problem formulation that contains fewer quadratic terms, and then generating valid inequalities that further tighten the lower bound provided by the problem relaxations solved at each node of the branch and bound tree. Chapter 6 is a discussion of the software contribution of this dissertation. In Chapter 7 we report the results of some computational experiments on both the heuristic and exact techniques, and in Chapter 8 we draw conclusions and make recommendations for further research.

1.3 Terminology and Symbols

The following tables provide a reference to the definitions, terms, and symbols used throughout this dissertation.
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>The set of nodes with junction node 0 and boundary nodes.</td>
</tr>
<tr>
<td>$n$</td>
<td>The number of CTM boundary nodes.</td>
</tr>
<tr>
<td>$K$</td>
<td>The set of observations or boundary conditions under which the electronic chip package is to be evaluated by the CTM.</td>
</tr>
<tr>
<td>$m$</td>
<td>The number of observations in the CTM.</td>
</tr>
<tr>
<td>$q$</td>
<td>The power input parameter.</td>
</tr>
<tr>
<td>$h_i^k$</td>
<td>The heat transfer coefficient at node $i = 1, \ldots, n$ for observation $k = 1, \ldots, m$.</td>
</tr>
<tr>
<td>$S_i$</td>
<td>The surface area represented by boundary node $i = 1, \ldots, n$.</td>
</tr>
<tr>
<td>$\tilde{T}_i^k$</td>
<td>The observed or reference temperature for node $i = 0, \ldots, n$ in observation $k = 1, \ldots, m$.</td>
</tr>
<tr>
<td>$T_A$</td>
<td>The ambient environmental temperature.</td>
</tr>
<tr>
<td>$w_i^k$</td>
<td>The relative importance weight of correctly estimating the temperature of node $i = 0, \ldots, n$ for observation $k = 1, \ldots, m$.</td>
</tr>
<tr>
<td>$r_{lb}, r_{ub}, r, \bar{r}$</td>
<td>Upper and lower bounds on $r$-variables, indexed by $i$ and $j$.</td>
</tr>
<tr>
<td>$T_{lb}, T_{ub}, T, \bar{T}$</td>
<td>Upper and lower bounds on $T$-variables, indexed by $i$ and $k$.</td>
</tr>
<tr>
<td>$h(\delta)$</td>
<td>The objective function of the CTM model with respect to the perturbation $\delta$ of a single thermal resistance value.</td>
</tr>
<tr>
<td>$g(\delta)$</td>
<td>A single term of the objective function of the CTM model with respect to the perturbation $\delta$ of a single thermal resistance value.</td>
</tr>
<tr>
<td>$\nu$</td>
<td>A parameter used in the UACTM weighting scheme for junction nodes.</td>
</tr>
<tr>
<td>$\nu'$</td>
<td>A parameter used in the UACTM weighting scheme for boundary nodes.</td>
</tr>
</tbody>
</table>
Table 1.2. Variable definitions.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_{ij}$</td>
<td>The resistance parameter value for link $(i,j)$, for $0 \leq i &lt; j \leq n$.</td>
</tr>
<tr>
<td>$r_{ij}$</td>
<td>The inverse of the resistance parameter $R_{ij}$.</td>
</tr>
<tr>
<td>$T_{ik}$</td>
<td>The predicted temperature for node $i = 0, \ldots, n$ under observation $k = 1, \ldots, m$.</td>
</tr>
<tr>
<td>$z$</td>
<td>The current objective function value.</td>
</tr>
<tr>
<td>$A_k$</td>
<td>A square symmetric matrix composed of given values for $r_{ij}$, $h_{ik}^k$, and $S_i$.</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>A variable used in the heuristic to substitute for the difference in certain terms of $A_k^{-1}$.</td>
</tr>
<tr>
<td>$F_k$</td>
<td>A nonsingular transformation matrix of the same dimensions as $A_k$ used in the heuristic.</td>
</tr>
<tr>
<td>$\Delta$</td>
<td>A perturbation matrix of the same dimensions as $A$.</td>
</tr>
<tr>
<td>$\Delta_k^i$</td>
<td>A variable used to translate $T_{ik}^k$ into a positive range, $T_{ik}^k = T_{ik}^k + \Delta_k^i$.</td>
</tr>
<tr>
<td>$\delta_{ij}$ or $\delta$</td>
<td>A perturbation of a variable $r_{ij}$.</td>
</tr>
<tr>
<td>$e_{ik}^j$</td>
<td>A variable used in the heuristic representing the current error between $T_{ik}^k$ and $\hat{T}_{ik}^k$.</td>
</tr>
<tr>
<td>$\overline{E}_k^i$</td>
<td>The amount by which $T_{ik}^k$ exceeds $\hat{T}_{ik}^k$.</td>
</tr>
<tr>
<td>$\underline{E}_k^i$</td>
<td>The amount by which $\hat{T}<em>{ik}^k$ exceeds $T</em>{ik}^k$.</td>
</tr>
<tr>
<td>$U_{ij}^k, V_{ij}^k$</td>
<td>Variables introduced to generate RLT relaxations of the CTM problem replacing the quadratic terms $\delta T$ or $\delta \Delta$.</td>
</tr>
<tr>
<td>$X_k^i$</td>
<td>Variable introduced in the RLT relaxation of the CTM.</td>
</tr>
<tr>
<td>$\tau_{ij}^k$</td>
<td>Variable introduced to reduce the number of quadratic terms in the CTM.</td>
</tr>
</tbody>
</table>

The units of temperature are in Kelvin, [K]. The surface areas, $S$, are in meters squared, [$m^2$]. The power input parameter, $q$, is in Watts, [W]. The heat transfer coefficients, $h$, are in units of Watts per meter squared - Kelvin, [W/m²(K)].

Table 1.3 describes the parameters used in the software implementation of heuristic and exact approaches to solving the CTM problem. This software is described in detail...
in Chapter 6. These parameters provide the user with choices whereby the software may be tuned to provide answers within a specified tolerance while seeking the most memory and CPU time efficient execution possible.

**Table 1.3.** Computation parameter definitions.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inner opt $\epsilon_1$</td>
<td>The termination $\epsilon$ criteria for the inner optimization step of the heuristic technique.</td>
</tr>
<tr>
<td>Inner opt count</td>
<td>The termination iteration criteria for the inner optimization step of the heuristic technique.</td>
</tr>
<tr>
<td>Bin bias</td>
<td>A constant used in the inner optimization technique for placing values into bins which permits a more efficient sort of these values.</td>
</tr>
<tr>
<td>Bisection $\epsilon_2$</td>
<td>The termination $\epsilon$ criteria for bisection search.</td>
</tr>
<tr>
<td>Bisection count</td>
<td>The termination iteration criteria for bisection search.</td>
</tr>
<tr>
<td>Confidence gap $g$</td>
<td>The amount by which to factor the gap between lowest and highest $r$-solution value in the heuristic to provide a starting range for the $r$-variables over which to optimize using RLT.</td>
</tr>
<tr>
<td>$h_{scale}$</td>
<td>A scale factor used to determine the probability with which to run the heuristic during the RLT algorithm.</td>
</tr>
<tr>
<td>RLT Type</td>
<td>The type of RLT relaxation to perform, equal to 0,1,2, or 3, corresponding to quadratic, linear, least absolute deviation, and revised quadratic formulations.</td>
</tr>
<tr>
<td>RRS Iteration count</td>
<td>The number of random restart iterations of the heuristic to perform.</td>
</tr>
<tr>
<td>RRS $r$ mean</td>
<td>The mean of the normal distribution used to generate random starting positions for the heuristic.</td>
</tr>
<tr>
<td>RRS $r$ s.d.</td>
<td>The standard deviation of the normal distribution used to generate random starting positions for the heuristic.</td>
</tr>
<tr>
<td>Tightness $\epsilon$</td>
<td>The $\epsilon$ used to determine whether a decision variable is tight against one or the other of its bounds in the branch and bound algorithm.</td>
</tr>
<tr>
<td>$depth$</td>
<td>The depth in the node list of a particular node, e.g. the number of arc in the node tree from a particular node to the root node.</td>
</tr>
</tbody>
</table>
These values are discussed in more detail in Chapter 6, and certain experiments are performed in Chapter 7 in order to develop some intuition on appropriate values for their practical use.
Chapter 2

COMPACT THERMAL MODELS

This chapter describes the compact thermal model and develops the optimization problem whose efficient solution permits the use of CTMs in practical application in the design of electronic components and systems. We begin with a discussion of the graph representation of the electronic chip package, and then demonstrate the way in which thermal behavior is modeled by inter-node relationships in the graph. We next discuss two formulations of an objective function and present the complete optimization problem. The chapter concludes with a comparison of the CTM to a general quadratically constrained optimization problem.

2.1 Electronic Chip Package

An electronic chip package is a component of an electronic device that houses and protects an embedded electronic chip, and provides access to the chip for power and data transfer through a grid of pins. The package is composed of various materials with different thermal and electrical conductivity and is soldered or socketed into a circuit board. The designer of an electronic device is concerned with the thermal behavior of the package, the board, and the device. As discussed in Chapter 1, the thermal behavior of an electronic chip package may be explored through the use of finite element models or through direct experimentation. In the CTM approach, we attempt to characterize the chip package through a set of thermal resistance values, so that the designer can obtain a reasonably accurate picture of the steady state thermal behavior of the package under an arbitrary set of boundary conditions. In order to accomplish this, the CTM provides a set of thermal resistance values that best characterize the package under a fixed, or reference, set of boundary conditions.
and their associated thermal responses. Once the thermal resistance values that best characterize the chip are known, the engineer has a reasonable and efficient model for predicting the thermal state the package will achieve under any set of boundary conditions that will be implemented in the design.

2.2 Graph Representation

The compact thermal model is a simplified network model of an electronic chip package. In a CTM, a graph, \( G \), contains a vertex set, \( N \) and an edge set, \( E \). We will label the vertices with indices \( i \) and \( j \) and number them \( 0, 1, \ldots, n \). Vertex zero will represent the junction node of the chip package. Vertices \( 1, \ldots, n \) will represent the boundary nodes. In the CTM, power is applied to the chip at the junction node, and heat radiates from the chip at the boundary nodes. Thermal communication among the nodes occurs along the edges in set \( E \). Any two nodes, \( i \) and \( j \) between which thermal communication is permitted by the CTM will be connected by an edge, \( E_{ij} \).

For the purpose of this discussion the terms “vertex” and “node” will be used equivalently, and we will use the term “edge” to imply that flow between nodes is not restricted to a particular direction.

As discussed in the literature review, various network topologies have been considered for the CTM. The simplest topology of interest is a two vertex network model in which the chip package is represented by a junction node, \( (0) \), and a single boundary node \( (1) \), which represents the surface of the entire chip. A diagram of this graph topology is given in Figure 2.1.
A second topology of interest is a star topology in which a single junction node (0) is connected to two or more boundary nodes (1, . . . , n) which are considered to be thermally isolated from one another. That is, no boundary node is adjacent to any other boundary node. Figure 2.2 depicts a diagram of this topology.

CTMs permit thermal communication between boundary nodes via edges which are called shunts. A diagram of a partially shunted model is shown at Figure 2.3.
We consider a fully shunted or clique topology in which each node is connected to every other node in this dissertation, as depicted in Figure 2.4. This assumption permits the optimization model to choose a restricted topology if such a topology is desirable, but does not restrict or artificially impose a condition in which nodes are presumed to be thermally isolated from one another. If, from an engineering perspective, it is desirable to assume a topology whose edge set is a proper subset of the full clique, the models presented here can be trivially modified to represent the assumption.
2.3 Thermal Behavior

The CTM assumes that an electronic chip package will reach a steady state temperature profile that takes the form of a temperature $T_i$ at each node $i = 0, \ldots, n$. These temperatures are an aggregated representation of the temperature profile of the physical regions of the chip package represented by particular nodes. Given environmental conditions like ambient temperature and air flow across the package, we assume that when power is applied to the electronic chip package at the junction node, the requirement for conservation of energy will drive the various regions of the chip to a steady state temperature. The temperature achieved at each boundary node will be a function of the surface area represented by that node, $S_i$, and a heat transfer coefficient at that node, $h_i$. The temperature achieved at the junction node will be a function of an input power parameter, $q$. Thermal communication between adjacent nodes occurs because conservation of energy is required. If two nodes are able to communicate with one another thermally (and with the ambient environment), then heat will flow between the nodes. The thermodynamic model used by the CTM stip-
ulates that the energy flow between two connected nodes, \( i \) and \( j \), is a function of their temperatures, \( T_i \) and \( T_j \), and a thermal resistance between the nodes, \( R_{ij} \). It is important to note that this thermal resistance is not the same as electrical resistance. The units of thermal resistance are degrees Kelvin per Watt \([\circ K/W]\). The energy relationship between two connected nodes, \( i \) and \( j \) is then given by

$$ e = \frac{T_i - T_j}{R_{ij}}. \quad (2.1) $$

### 2.4 Thermal Balance in a CTM

Given the topologies and thermal behavior definitions from above, it is now possible to describe the thermal behavior of an electronic chip package under these assumptions. Figure 2.5 shows the energy input, output, and flow within a small CTM representation.

![Thermal diagram of a CTM](image)

**Figure 2.5.** Thermal diagram of a CTM.

The requirement for energy conservation enforces a conservation equation at each of the nodes. In the example given in Figure 2.5, the CTM enforces the following
condition at the junction node:

\[-q = \frac{T_1 - T_0}{R_{01}} + \frac{T_2 - T_0}{R_{02}} + \frac{T_3 - T_0}{R_{03}}.\]  

(2.2)

The three boundary nodes provide the three equations:

\[h_1S_1T_1 = \frac{T_0 - T_1}{R_{01}} + \frac{T_2 - T_1}{R_{12}} + \frac{T_3 - T_1}{R_{13}},\]  

(2.3a)

\[h_2S_2T_2 = \frac{T_0 - T_2}{R_{02}} + \frac{T_1 - T_2}{R_{12}} + \frac{T_3 - T_2}{R_{23}},\]  

(2.3b)

\[h_3S_3T_3 = \frac{T_0 - T_3}{R_{03}} + \frac{T_1 - T_3}{R_{13}} + \frac{T_2 - T_3}{R_{23}}.\]  

(2.3c)

In general for an electronic chip package modeled by \(N\) nodes, \(0, \ldots, n\), we have the equations:

\[-q = \sum_{j=0}^{n} \frac{T_j - T_0}{R_{0j}},\]  

(2.4a)

\[h_iS_iT_i = \sum_{j \in \{0, \ldots, n\} \setminus i} \frac{T_i - T_j}{R_{ij}} \quad \forall i = 1, \ldots, n.\]  

(2.4b)

### 2.5 Varying the Boundary Conditions

As discussed in Chapter 1, the purpose of the CTM is to provide a thermal characterization of a particular electronic chip package under a wide range of conditions. This provides the electronic system designer a way to accurately determine the anticipated behavior of a subassembly in operation under the boundary conditions provided by the design. In order to account for a range of reference boundary conditions in the model, we introduce the index \(k\) to represent a particular condition from the set of modeled boundary conditions \(1, \ldots, m\). In a CTM of a particular chip package we will hold the geometry constant, i.e., the surface area of a given node \(S_i\) will remain constant for each boundary condition observation \(k\). The input energy parameter, \(q\) will likewise remain the same for each observation. The design of a CTM requires the
specification of a set, \( K \), of boundary conditions for each of the heat transfer coefficients \( h_i, i = 1, \ldots, n \). The CTM finds steady state temperatures at each node under each of these boundary conditions, and given a choice of \( R_{ij} \) for each edge \((ij)\). These resistance values remain fixed for all observations. Therefore, the equations given by (2.4a) are written in the case where the number of observations, \( m \), is greater than one by using an index \( k \) where needed as

\[
-q = \sum_{j=0}^{n} \frac{T_j^k - T_0^k}{R_{0j}}, \quad \forall k = 1, \ldots, m, \tag{2.5a}
\]

\[
h_i^kS_iT_i^k = \sum_{j \in \{0, \ldots, n\}} \frac{T_i^k - T_j^k}{R_{ij}} \quad \forall i = 1 \ldots n, \quad \forall k = 1, \ldots, m. \tag{2.5b}
\]

### 2.6 Objective of the CTM

The system designer would like to identify the set of resistance values, \( R_{ij} \), that best characterize the interaction between regions of the chip package under a wide selection of boundary conditions. Therefore the objective function that the CTM attempts to optimize is a measure of the degree to which the temperature predicted at each node 0, \ldots, \( n \) and under each boundary condition 1, \ldots, \( K \) differs from a reference temperature, \( \hat{T}_i^k \). These reference temperatures may be derived by direct experimentation on a similar chip package or from the use of a fully detailed model. The CTM also permits the weighting of these differences, since it is typically desirable to match the junction node temperatures to their reference values with greater fidelity than that of the boundary nodes. We examine two objective function forms in this paper. The first is a weighted least squares deviation and the second is a weighted least absolute deviation. These are given in Equations (2.6a) and (2.6b):

\[
z = \sum_{i=0}^{n} \sum_{k=1}^{m} w_i^k \left( \hat{T}_i^k - T_i^k \right)^2 \tag{2.6a}
\]

\[
z = \sum_{i=0}^{n} \sum_{k=1}^{m} w_i^k |\hat{T}_i^k - T_i^k|. \tag{2.6b}
\]
It is worth noting that each of these functions is a convex, nonlinear function of the $T$-variables. The selection of appropriate weights is worth consideration. The modeling purpose of weights that vary by node and boundary condition is to permit an objective function that gives greater weight to matching the predicted and reference temperatures of the junction node with a greater preference than the boundary nodes. One existing weighting scheme used in the UACTM model calculates the weights by node and observation according to the following two equations. Equation (2.7a) is used to calculate junction node weights and Equation (2.7b) is used to calculate boundary node weights:

\[
\begin{align*}
    w_k^0 &= \frac{Y}{(T_k^0)^2m} \quad \forall k = 0, \ldots, m, \\
    w_i^k &= \frac{W(h_i^k)^2(S_i)^2}{mnq} \quad \forall i = 1, \ldots, n \quad \forall k = 0, \ldots, m.
\end{align*}
\]

(2.7a) (2.7b)

In these equations the variables are as defined in Table 1.1. Because of the units and dimensions of typical problem data, this tends to yield poorly scaled problem weights. In [63] we used this weighting scheme in order to make comparisons between the heuristic described in Chapter 3 and the existing UACTM method. However, a better scaled weighting scheme that still places appropriate priority on the junction node was adopted for testing that did not require this comparison. This scheme gives the junction node a weight of $n$ in each observation and each boundary node a weight of 1.0 for each observation.
2.7 The CTM Optimization Model

Given the objective and energy balance constraints, the optimization problem described by a CTM is given by

\[
\text{Minimize } z \quad (2.8a)
\]
\[
\text{s.t. } \sum_{j=1}^{n} \frac{T_j^k - T_0^k}{R_{0j}} = -q \quad \forall k = 1, \ldots, m \quad (2.8b)
\]
\[
\sum_{j \in \{0, \ldots, n\} \setminus i} \frac{T_j^k - T_i^k}{R_{ij}} = h_i^k S_i T_i^k \quad \forall i = 1, \ldots, n, \ \forall k = 1, \ldots, m, \quad (2.8c)
\]

where \( z \) is given by either of the objectives (2.6a) or (2.6b). In Chapter 3 we develop a heuristic technique for solving the quadratic case. In later chapters we develop global optimization methods that make use of each of these objectives. In either case the objective is convex, while the constraint set given by (2.8b) and (2.8c) is nonconvex.

In implementation it is often useful to make the substitution \( R_{ij} = 1/r_{ij} \) where the variables \( r_{ij} \) are understood to be restricted from taking on a value strictly equal to zero.

2.8 The CTM as a General Quadratic Programming Problem

The CTM problem with quadratic objective can be written in the form of a general (nonconvex) quadratic programming problem (QPP) with quadratic constraints. Thus it is open to solution by those techniques reviewed in Chapter 1. To write a CTM as a QPP we allow the substitution \( R_{ij} = 1/r_{ij} \) and write the variables as a vector \( x = (r, T_1, \ldots, T_k) \). Then the problem may be written in matrix form as

\[
\text{Minimize } xQx + Dx + C \quad (2.9a)
\]
\[
\text{s.t. } xV_i x + A_i x = b \quad \forall i = 0, \ldots, n. \quad (2.9b)
\]
In this representation the matrix $Q$ is diagonal with zero entries corresponding to the variables $r_{ij}$ and with the weights $w_i^k$ on the remaining diagonal elements. The constant $C$ represents the sum of the weighted squares of the reference temperatures. The matrices $V$ and $A$ generate the nonconvex constraint set. The right hand side vector $b$ is a zero vector except in those constraints corresponding to the junction node whose element is equal to $-q$. This formulation does not exploit the separability of the decision variable vector into $m$ separate vectors, $x_k = (r, T_k)$, since the complicating variables $r$ do not permit the solution of the CTM as $m$ separate sub-problems in any case. The general quadratic programming problem is known to be a hard problem, and hence we seek methods of solving our special case that do not rely on general purpose solution techniques for solving general quadratic problems. We may hold out some hope that the special structure of the CTM will permit its solution to global optimality in less time than an instance of a general quadratic programming problem of similar dimensions would require.

2.9 CTM Data

The data given in the CTM model are defined in Chapter 1 in Table 1.1. The reference temperatures are in degrees Kelvin. The UACTM model calculates a reference temperature from ambient temperature and other problem data using the following two equations:

$$\hat{T}_0^k - T_A = q \quad \forall k = 1, \ldots, m$$

$$\widehat{T}_i^k - T_A)S_i = h_i^k \quad \forall i = 0, \ldots, n \quad \forall k = i, \ldots, m$$

where $T_A$ is the ambient environmental air temperature.

The design of the boundary conditions, $h_i^k$ for $i = 0, \ldots, n$ and for $k = 1, \ldots, m$ is a topic beyond the scope of this paper, but the CTM data we explore in this paper consists of either 9 or 99 observations and a smaller number of nodes, 5, 6, or 7.
HEURISTIC TECHNIQUES

In this chapter we develop a heuristic approach to solving the CTM problem. The techniques described in this chapter are presented in [63]. While this heuristic scheme does not necessarily provide a globally optimal solution, it does give useful feasible solutions at relatively modest computational expense by using the principle of coordinate descent and an efficient inner optimization scheme. We first discuss some of the special structure of the problem and then exploit these observations to build the heuristic.

3.1 Some Observations on the CTM Problem

The Quadratic CTM problem was given in Chapter 2 as

\[
\begin{align*}
\text{Minimize } & z = \sum_{i=0}^{n} \sum_{k=1}^{m} w_i^k \left( \tilde{T}_i^k - T_i^k \right)^2 \\
\text{s.t. } & \sum_{j=1}^{n} \frac{T_j^k - T_0^k}{R_{0j}} = -q \quad \forall k = 1, \ldots, m \quad (3.1b) \\
& \sum_{j \in \{0, \ldots, n\} \setminus i} \frac{T_j^k - T_i^k}{R_{ij}} = h_i^k S_i T_i^k \\
& \forall i = 1, \ldots, n, \forall k = 1, \ldots, m \quad (3.1c)
\end{align*}
\]

If we fix the variables \(r_{ij} = 1/R_{ij}\) to some set of feasible values, then the constraint set given by (3.1b) and (3.1c) can be written as a set of \(m\) systems of linear equations \(A_k t_k = b\) where \(t_k = (T_0^k, \ldots, T_n^k)^\top\). Dropping the index \(k\) for clarity and examining a three node case, each of these systems takes the form

\[
\begin{pmatrix}
-r_{01} - r_{02} & r_{01} & r_{02} \\
r_{01} & -r_{01} - r_{12} - h_1 S_1 & r_{12} \\
r_{02} & r_{12} & -r_{02} - r_{12} - h_2 S_2
\end{pmatrix}
\begin{pmatrix}
T_0 \\
T_1 \\
T_2
\end{pmatrix}
= \begin{pmatrix}
-q \\
0 \\
0
\end{pmatrix}.
\] (3.2)
In general, the matrix $A_k$ is composed of elements $a_{ij}^k$ as defined here

$$a_{00}^k = \sum_{j=1}^{n} -r_{0j}$$  \hspace{1cm} (3.3a)

$$a_{ii}^k = -h_i s_i - \sum_{j \in (0, ..., n \setminus i)} r_{ii} \ \forall i = 1, ..., n$$  \hspace{1cm} (3.3b)

$$a_{ij}^k = r_{ij} \ \forall i = 0, ..., n \ \forall j = 0, ..., n \ i \neq j$$  \hspace{1cm} (3.3c)

Our heuristic technique starts from an initial feasible guess, $r$, and the corresponding solution to (3.2), $t$. It then proceeds to repeatedly optimize the objective function (3.1a) by successively choosing perturbations of individual values of $r_{ij}$, while holding all other $r$-values constant. This is an application of the optimization technique known as coordinate descent as described in [17] where the values of all but one decision variable are held constant and the objective is minimized by choice of a perturbation of the remaining decision variable.

### 3.2 Perturbing a Single Value of $r_{ij}$

Let us consider an initial feasible guess for $r$ and solutions $\tilde{t}_k$ for the $m$ systems of equations whose coefficient matrices follow the pattern given in (3.3)

$$\tilde{t}_k = A_k^{-1}b.$$  \hspace{1cm} (3.4)

We can make use of the rank one update technique described in, for example [31], in the following manner. If we make a perturbation, $\delta$, of a single $r_{uv}$, then we perturb each matrix $A_k$ and must solve $m$ new system of equations

$$\tilde{A}_k t = b$$  \hspace{1cm} (3.5)

where

$$\tilde{A}_k = A_k + \Delta.$$  \hspace{1cm} (3.6)
The matrix $\Delta$ is of the same dimension as $A$ and has exactly four nonzero elements, $\delta_{uv}$. These are

$$\delta_{uu} = -\delta$$

$$\delta_{vv} = -\delta$$

$$\delta_{uv} = \delta$$

$$\delta_{vu} = \delta.$$ (3.7a)

In the following discussion we ignore the subscript $k$ for clarity. Rather than resorting to a naive technique for solving these systems (3.6) (e.g. $LU$ decomposition and back substitution as described in [64]), we employ a nonsingular transformation matrix $F$ defined so that $AF = \overline{A}$. Since $AF = \overline{A}$, $A$ is known and nonsingular and $A^{-1}$ has already been calculated, we know that $F = A^{-1}\overline{A}$. Let $a_{ij}^{-1}$ be the elements of $A^{-1}$. Then $F = A^{-1}(A + \Delta) = I + A^{-1}\Delta$ (where $I$ is an identity matrix of appropriate dimension) and the elements of $F$ are given by

$$f_{ij} = 0 \text{ for } i \in \{0, \ldots, n\}, \ j \in \{0, \ldots, n\} \setminus \{u, v, i\}$$

$$f_{ii} = 1 \text{ for } i \in \{0, \ldots, n\} \setminus \{u, v\}$$

$$f_{iu} = \delta(a_{iv}^{-1} - a_{iu}^{-1}) \text{ for } i = 0, \ldots, n, i \neq u$$

$$f_{uu} = 1 + \delta(a_{uv}^{-1} - a_{uu}^{-1})$$

$$f_{iv} = -\delta(a_{iv}^{-1} - a_{iu}^{-1}) \text{ for } i = 0, \ldots, n, i \neq v$$

$$f_{vv} = 1 - \delta(a_{uv}^{-1} - a_{uu}^{-1}).$$ (3.8a)

If we define $\alpha_i = a_{iv}^{-1} - a_{iu}^{-1}$, then we can solve the system of equations $Ft = \overline{t}$ for individual values $T_i$ given the values $\overline{T}_i$ in the current solution by the following approach. First solve the system

$$(1 + \delta\alpha_u)T_u - \delta\alpha_u T_v = \overline{T}_u$$

$$\delta\alpha_v T_u + (1 - \delta\alpha_v)T_v = \overline{T}_v.$$ (3.9a)
for $T_u$ and $T_v$. Note that this system has a unique solution if and only if $\det(F) = 1 + \delta (\alpha_u - \alpha_v) \neq 0$. This requires a careful selection of $\delta$ at each step to ensure that the transformation matrix is full rank. Given that $\det(F) \neq 0$ the solution to (3.9) is given by

$$T_u = \frac{(1 + \alpha_v) T_u + \delta \alpha_u T_v}{\det(F)}$$  \hspace{1cm} (3.10a)

$$T_v = \frac{(-\delta \alpha_v) T_u + (1 + \delta \alpha_u) T_v}{\det(F)}.$$  \hspace{1cm} (3.10b)

For $i \neq u, v$ the values of $T_i$ are given by

$$T_i = \bar{T}_i + \delta \alpha_i (T_v - T_u) = \bar{T}_i + \delta \alpha_i \frac{(T_v - T_u)}{\det(F)}.$$  \hspace{1cm} (3.11)

### 3.3 Effect of a Small Perturbation on the Objective Function

In the previous section we demonstrated a technique for determining the change to the predicted temperatures $T_i^k$, $i = 0, \ldots, n, k = 0, \ldots, m$ given the perturbation of a single $r$-value by an amount $\delta$. In this section we demonstrate how this closed form expression for the updated $T$-values permits us to describe the change in the quadratic objective function of the CTM as a function of such small perturbations.

Let us define $e_i^k$ as the error in the current predicted temperature, $\bar{T}_i^k$ relative to the reference temperature $\hat{T}_i^k$, $(e_i^k = \bar{T}_i^k - \hat{T}_i^k)$. Then under a perturbation of resistance value $r_{uv}$ by the amount $\delta$, the quadratic objective function becomes

$$z(\delta) = \sum_{k=1}^{m} \left[ \sum_{i \neq u, i \neq v} w_i^k \left( e_i^k + \delta \left( \frac{\alpha_u^k \left( \alpha_v^k \left( T_v^k - \hat{T}_u^k \right) - \alpha_u^k \left( \bar{T}_v^k - \bar{T}_u^k \right) \right)}{\det(F)} \right) \right)^2 + \left( \frac{\alpha_u^k \left( \alpha_v^k \left( T_v^k - \hat{T}_u^k \right) - \alpha_u^k \left( \bar{T}_v^k - \bar{T}_u^k \right) \right)}{\det(F)} \right)^2 + \left( \frac{\alpha_v^k \left( \alpha_u^k \left( \bar{T}_v^k - \bar{T}_u^k \right) \right)}{\det(F)} \right)^2 \right].$$  \hspace{1cm} (3.12)
For simplicity, let us rewrite the function as

\[ h(\delta) = \sum_{i=0}^{n} \sum_{k=1}^{m} \left( \frac{\lambda_i^k + \mu_i^k \delta}{1 + \sigma^k \delta} \right)^2, \quad (3.13) \]

where we have taken

\[ \lambda_i^k = \sqrt{w_i^k e_i^k} \quad (3.14a) \]

\[ \mu_i^k = \begin{cases} \sqrt{w_i^u} \left( \alpha_i^k \left( T_v^k - \tilde{T}_v^k \right) - \alpha_i^k \left( \tau_u^k \right) \right) & \text{if } i = u \\ \sqrt{w_i^v} \left( \alpha_i^k \left( \tau_v^k \right) + \alpha_i^k \left( \tilde{T}_v^k - T_u^k \right) \right) & \text{if } i = v \\ \sqrt{w_i^i} \left( \tau_i^k \left( \alpha_i^k - \alpha_i^k \right) + \alpha_i^k \left( \tilde{T}_v^k - T_u^k \right) \right) & \text{else} \end{cases} \quad (3.14b) \]

\[ \sigma^k = \alpha_u^k - \alpha_v^k. \quad (3.14c) \]

### 3.4 Optimizing the Perturbed Objective Function

The function \( h(\delta) \) developed in the previous two sections is nonconvex, has a non-negative range, and is undefined at points \( \delta = -1/\sigma^k \) for each \( k = 1, \ldots, m \) such that \( \sigma^k \neq 0 \). This function is twice differentiable wherever it is defined.

In this section we analyze the function \( h(\delta) \) in order to develop a fast heuristic technique for finding a good quality minimizing value of \( \delta \). This inner-optimization step of the overall heuristic should quickly find a near optimal value for the perturbation, \( \delta \). Our technique for optimizing this function proceeds by partitioning a suitably bounded domain into sub-regions and undefined points, and then searching these sub-regions in an efficient manner for a near optimal solution. In order to best understand the procedure we provide some analysis of \( h(\delta) \).

#### 3.4.1 Analysis of \( h(\delta) \)

We label the individual terms of (3.13) corresponding to a particular combination of \( i \) and \( k \) as \( g(\delta) \), and again drop the indices for clarity. The first and second derivatives
of $g(\delta)$ are given by:

$$\frac{\partial g}{\partial \delta} = \frac{2(\mu - \lambda \sigma)(\lambda + \mu \delta)}{(1 + \sigma \delta)^3} \quad \text{and} \quad (3.15a)$$

$$\frac{\partial^2 g}{\partial \delta^2} = 2(\mu - \lambda \sigma) \left[ \frac{\mu - 3\sigma \lambda - (2\sigma \mu) \delta}{(1 + \sigma \delta)^4} \right]. \quad (3.15b)$$

These derivatives imply that the individual terms have at most a single undefined point, a single extreme point (minimum), and a single inflection point. If these points are all defined and distinct, then the individual term takes the form shown in either Figure 3.1 or Figure 3.2. Specifically if $1/\sigma < \lambda/\mu$ the functional form will be as in

![Figure 3.1](image)

**Figure 3.1.** Plot of $g(\delta)$ when $1/\sigma < \lambda/\mu$.

Figure 3.1 and as in Figure 3.2 otherwise. In either case the function is nonnegative everywhere and undefined at the point $\delta = -1/\sigma$. If $\mu \neq 0$, the term will achieve a minimum of zero at the single root of the first derivative,

$$\delta = -\frac{\lambda}{\mu}, \quad (3.16)$$

and if both $\sigma \neq 0$ and $\mu \neq 0$, it will have an inflection point at the single root of the second derivative at

$$\delta = \frac{\mu - 3\sigma \lambda}{2\sigma \mu}. \quad (3.17)$$
This term approaches the limit \((\mu/\sigma)^2\) from above as \(\delta\) goes to either plus or minus infinity, and the same limit from below as \(\delta\) goes to the opposite extreme. There also exists the possibility that the term will take on one of four special cases that arise when the points discussed above coincide or are not defined. These cases are enumerated here.

**Case 1:** If \(\sigma = \mu = 0\), then the term describes the line \(g(\delta) = \lambda^2\), and if in addition \(\lambda = 0\) then the line is \(g(\delta) = 0\).

**Case 2:** If \(\sigma = 0\) and \(\mu \neq 0\), then the term describes the parabola \(g(\delta) = (\lambda + \mu \delta)^2\), with a minimum of zero at \(\delta = -\lambda/\mu\).

**Case 3:** If \(\sigma \neq 0\) and \(\mu = 0\) with \(\lambda \neq 0\), then the second derivative of \(g(\delta)\) given by (3.15b) is always positive, and the term takes on the shape shown in Figure 2. In this case the term approaches \(+\infty\) at the undefined point, and zero as \(\delta\) goes to \(\pm\infty\).
Case 4: If $\sigma \neq 0$, and either $\mu = \lambda = 0$, or $\lambda \neq 0$ and $\sigma = \mu/\lambda$, then $g(\delta) = \lambda^2$ for $\delta \neq -1/\sigma$, and is undefined at $\delta = -1/\sigma$. Figure 3.4 provides an example of this case.

Note that there exists the possibility of an inflection point at some $\delta$ in any individual term $g(\delta)$. This means that we can make no general statement about the convexity of $h(\delta)$ over a broad range of $\delta$, nor can we guarantee the convexity of $h(\delta)$.
between any two adjacent undefined points. This makes finding a global minimum for \( h(\delta) \) difficult, and rather than seeking a global minimum, we will use the structure of the individual terms to quickly find a good quality local minimum.

Given appropriate upper and lower bounds on the domain of \( \delta \), we can divide the domain of \( \delta \) into no more than \( m + 1 \) disjoint intervals in which the function is continuous and differentiable, where the intervals are separated by undefined points \( -1/\sigma^k \) for \( k = 1, \ldots, m \) such that \( \sigma^k \neq 0 \). Next we can partition these intervals into sub-regions so that within a given sub-region, each individual term \( g(\delta) \) is either non-increasing or nondecreasing. In a particular sub-region it is possible that some terms are nonincreasing and others are nondecreasing, and no knowledge of the convexity of an individual term or of the function is necessary. For each of the sub-regions we establish a lower bound on \( h(\delta) \) by summing each term’s lower bound over the sub-region. These term-wise lower bounds must occur at one or the other of the region boundaries. The sub-region lower bound calculation is illustrated in Figure 3.5.

\[ \begin{array}{c}
\text{Figure 3.5. Example of calculating region lower bounds.}
\end{array} \]

Figure 3.5 shows three \( g(\delta) \) terms superimposed with their partition points marked.
Note that in the sub-region marked \(a\), the lower bound in the sub-region is the sum of the lower bounds on each term, which do not occur at the same boundary, while in the region marked \(b\), each term is decreasing, and the region lower bound equals the minimum of \(h(\delta)\) achieved at the rightmost point in the sub-region. Within each sub-region the function is continuous and differentiable, which implies that at least one local minimum must exist in each sub-region. However, since we have no guarantee of convexity in a sub-region, multiple local minima are possible. We test for the condition shown in the region labeled \(b\) in Figure 3.5, which immediately provides a minimum for \(h(\delta)\) in that sub-region. If this condition does not exist, we employ bisection search, as described in [49], to provide a local minimum on the interior of the sub-region. The quality of this minimum is known relative to the lower bound on the sub-region, and in practice, the conditions that permit the existence of multiple minima in a sub-region are rare. Although we could seek a global minimum over the sub-region, by using for example a finite element search, the additional computation effort to do so appears to outweigh any potential solution quality benefits.

### 3.4.2 Inner optimization algorithm

Our inner optimization algorithm is executed in five steps: calculating sub-region boundary points, sorting these boundary points to form a partition of sub-regions, bounding \(h(\delta)\) from below in the sub-regions, sorting the sub-regions by these bounds, and searching the sub-regions.

**Step 1:** Calculate the problem’s “boundary points.” We define boundary points of two types. The first type occurs at the \(m\) undefined points, \(\delta = -1/\sigma^k\). The second type occurs when the slope of any individual term changes sign at \(\delta = -\lambda/\mu\), the only root of the first derivative of an individual term. We append appropriately defined upper and lower bounds on \(\delta\) that contain all of the boundary points to the list. This provides as many as \(m + nm + 2\) distinct boundary points. During the
calculation of these boundary points we account for the special cases described in Section 3.4.1, and for cases in which duplicate boundary points are found. Further, our algorithm accounts for the possibility that a boundary point is not contained in the current domain bound, expanding the bounding values as necessary. In this step we also record an incumbent upper bound of \( h(0) \) which is just the value of the objective function if no \( r \)-value is perturbed from the current solution.

**Step 2:** Sort the boundary points. For efficiency, we first place the boundary points that partition the domain of \( \delta \) in a set of bins as they are calculated, then sort each bin, and then finally generate a single sorted list of boundary points with the lower bound on \( \delta \) as its first element and the upper bound as its last. This sorted list of boundary points defines \( m + nm + 1 \) regions over which we search for a value of \( \delta \) that minimizes \( h(\delta) \). (This excludes the regions beyond the bounding values of \( \delta \), which inhibits very large changes in \( r \)-value in a single descent step of the outer optimization.)

Rather than naively search each region in Steps 3 through 5, we use an implicit enumeration of the minima in each region, and terminate when there is no possibility of finding a better solution than the current incumbent.

**Step 3:** Determine the lower bound in each region. This step follows the procedure illustrated in Figure 3.5. For each of the regions, we examine each term of (3.13), and determine the slope of the term’s first derivative at the region midpoint. If the slope is positive, then the term achieves a minimum at the left bound of the region, otherwise the minimum is at the right bound. The sum of these term-wise minima provides a lower bound for (3.13) over that region. If all of the slopes happen to be of the same sign, then this lower bound is in fact equal to the minimum objective over the region. We calculate these lower bounds for each sub-region, and place them in a search list only if they are less than the incumbent upper bound calculated in Step 1.

**Step 4:** Sort the search list in nondecreasing order of their lower bounds. The least of these lower bounds is an absolute lower bound for the global minimum of
(3.13) within the bounded domain.

**Step 5:** Starting with the sub-region that has the smallest lower bound, use bisection search to find a local minimum. If this solution is better than the incumbent, update the incumbent solution and proceed to the next sub-region on the search list. While the lower bound of the next sub-region on the search list is less than the incumbent objective value, use bisection search on the next sub-region on the search list to find a local minimum. Once the lower bound of the next sub-region on the search list is at least as large as the incumbent objective, or when the search list has been exhausted, terminate the algorithm and return the incumbent solution.

### 3.4.3 Overall heuristic algorithm

In this section we describe the overall heuristic algorithm in pseudo-code and discuss its use in a random restart algorithm.

A pseudo-code description of the overall heuristic is given here. For this implementation, we stop the heuristic after an examination of all $R_{ij}$-values does not appreciably decrease the objective function, or after a predetermined limit of sweeps through each $R_{ij}$-value is reached. For the former criterion, we measure the relative objective change as $\frac{(\text{old objective value})-(\text{new objective value})}{(\text{new objective value})}$, and stop once this value is smaller than some parameter $\epsilon_1$.

**Initialization:** Set the $R$-values to an initial guess, and solve the $m$ systems of linear equations to obtain the temperature estimates under each observation given the initial guess. This step provides an initial feasible solution and an incumbent objective function value. Set $\text{done} := \text{false}$.

**While** $(\text{done} == \text{false})$

$\text{done} := \text{true}$

**For** $(i = 0 \text{ to } n - 1)$

$\text{For} \ (j = i + 1 \text{ to } n)$
Select $R_{ij}$ for inner optimization
Calculate the values for $\lambda$, $\mu$, and $\sigma$ given in (3.14)
Compute $\delta = \min h(\delta)$ by the steps given in Section 3.4.2
Update solution $\overline{T}$ using (3.10), and incumbent objective using (3.12)

End For

End For

If (relative change to objective $> \epsilon_1$) then set $done := false$
If (iteration limit reached) then set $done := false$

End While

Return the decision variable values, $R$ and $T$.

Since the heuristic performs a coordinate descent from a fixed starting position, it is appropriate to examine its use as a step within a random restart algorithm. This approach would randomly generate starting solutions for $R_{ij}$-values, and execute the foregoing heuristic from each such set of initial $R$-values. The final solution corresponds to the best overall solution found during this process. Key questions that arise from this process are (a) how many iterations are required of this random search procedure before additional restarts cease to promise improvements on solution quality, and (b) whether it is appropriate for additional computational time to be spent on more restarts or due to stricter tolerance values in the heuristic itself. In Chapter 7 we examine the trade-off between various parameters in a random restart heuristic approach.

3.4.4 Notes on the heuristic

The heuristic algorithm developed in this chapter has proven useful in augmenting existing approaches for solving the CTM optimization problem. Existing techniques are iterative and require user interaction, while this method offers the promise of a hands-off generation of good quality resistance values applying the CTM to elec-
tronic system design. Moreover, the heuristic developed in this chapter is critical to providing the exact techniques developed in later chapters with high quality upper bounds. In Chapter 7 we also examine the value of the heuristic compared to a global optimization approach. In [63] we make comparisons between the heuristic and the iterative and interactive use of the generalized reduced gradient nonlinear solver provided in Microsoft Excel, which is analogous to the procedure described by DeVoe and Ortega in the UACTM model.
Chapter 4

AN RLT APPROACH

This chapter describes the application of the Reformulation Linearization Technique, described by Sherali and Adams in [55], to the CTM optimization problem. The RLT scheme is a convexification/partitioning algorithm that provides a globally optimal solution to the CTM model. Our application makes use of the heuristic developed in Chapter 3 and represents the first known application of a global technique to solving this problem.

4.1 RLT

The Reformulation Linearization Technique is applicable to problems whose objectives and/or constraints are of arbitrary (including rational) polynomial degree. RLT proceeds by solving a sequence of convex relaxations of the original problem. The relaxations are enumerated in a binary tree structure in which branching induces the partition of the domain of a particular variable into two regions. Ideally, the algorithm does not need to enumerate all such possible partitions because the tree structure may be pruned, as in integer branch and bound, by infeasibility and by bound. Additionally, if a solution at a particular node of the tree is feasible to the original problem, the node may be fathomed. This occurs when every branching variable in the sub-problem at a particular node is tight against either its current upper or lower bound. The following discussion illustrates the technique.

Consider a problem that contains a term in its formulation that is the product of two variables, \(XY\), and let these variables have bounds, \(X_l \leq X \leq X_u\) and \(Y_l \leq Y \leq Y_u\). We perform a linearization on this second order term by introducing a variable \(W\), which replaces \(XY\) in the original problem, and by adding the following
constraints, which are a consequence of the bounds on $X$ and $Y$. The bounds result in these nonnegative relationships.

\[
\begin{align*}
X_u - X &\geq 0 \quad (4.1a) \\
X - X_l &\geq 0 \quad (4.1b) \\
Y_u - Y &\geq 0 \quad (4.1c) \\
Y - Y_l &\geq 0. \quad (4.1d)
\end{align*}
\]

Cross multiplying these factors produces inequalities that contain a term $XY$.

\[
\begin{align*}
(X_u - X)(Y_u - Y) &\geq 0 \quad (4.2a) \\
(X_u - X)(Y - Y_l) &\geq 0 \quad (4.2b) \\
(X - X_l)(Y_u - Y) &\geq 0 \quad (4.2c) \\
(X - X_l)(Y - Y_l) &\geq 0. \quad (4.2d)
\end{align*}
\]

These imply the following constraints on $W$:

\[
\begin{align*}
W &\geq Y_l * X + X_l * Y - X_l * Y_l \quad (4.3a) \\
W &\geq Y_u * X + X_u * Y - X_u * Y_u \quad (4.3b) \\
W &\leq Y_u * X + X_l * Y - Y_u * X_l \quad (4.3c) \\
W &\leq Y_l * X + X_u * Y - Y_l * X_u. \quad (4.3d)
\end{align*}
\]

These constraints are a relaxation of the relationship $W = XY$. If either $X$ or $Y$ equals to its upper or lower bound, then $W = XY$ holds at equality, as two of the
constraints (4.3) will hold at equality. As a concrete example consider:

\[0 \leq X \leq 1\]  \hspace{1cm} (4.4a)
\[0 \leq Y \leq 1\]  \hspace{1cm} (4.4b)
\[X = 0.5\]  \hspace{1cm} (4.4c)
\[Y = 0.5\]  \hspace{1cm} (4.4d)
\[XY = 0.25\]  \hspace{1cm} (4.4e)
\[W \geq 0\]  \hspace{1cm} (4.4f)
\[W \geq 0\]  \hspace{1cm} (4.4g)
\[W \leq 0.5\]  \hspace{1cm} (4.4h)
\[W \leq 0.5\]  \hspace{1cm} (4.4i)

At the current solution, \((X, Y) = (0.5, 0.5)\), neither of the variables are tight against a current bound. The variable \(W\) is not constrained to be equal to the current value of \(XY\). If we chose to branch on the variable \(X\) at its current solution, we generate two child nodes, one in which \(0 \leq X \leq 0.5\) and one in which \(0.5 \leq X \leq 1\). Now let the solution to the first of these be:

\[0 \leq X \leq 0.5\]  \hspace{1cm} (4.5a)
\[0 \leq Y \leq 1\]  \hspace{1cm} (4.5b)
\[X = 0.5\]  \hspace{1cm} (4.5c)
\[Y = 0.5\]  \hspace{1cm} (4.5d)
\[XY = 0.25\]  \hspace{1cm} (4.5e)
\[W \geq 0\]  \hspace{1cm} (4.5f)
\[W \geq 0.25\]  \hspace{1cm} (4.5g)
\[W \leq 0.5\]  \hspace{1cm} (4.5h)
\[W \leq 0.25\]  \hspace{1cm} (4.5i)
In this case the condition $W = XY$ holds by the constraints on $W$, and the solution to the relaxation in which $XY$ has been replaced by $W$ is feasible to a problem in which no such relaxation has been made.

The RLT implicitly enumerates, by use of a branch and bound tree, the set of convex relaxations of the original problem. As in integer branch and bound [76], nodes may be fathomed by bound and infeasibility. In a fashion analogous to fathoming by integrality, a node may also be fathomed by the condition in which all substitute variables are strictly equal to the second order term they represent. When this condition holds true, the solution is a feasible solution to the original problem and a candidate for updating the current upper bound. We call such solutions “nonlinear-feasible” or “tight” for short. Additionally, the branch and bound algorithm need not search this node further for a better solution to the original problem. The RLT method is known to terminate finitely or to result in a tree of infinite length with the property that any accumulation point on a path of infinite length will be a globally optimal solution. The proof of this property is provided in [55].

4.2 Three Relaxations of CTM

The CTM described in Chapter 2 consists of either a quadratic or least absolute deviation objective function and a nonconvex set of quadratic constraints. In this section we describe three potential convex relaxations that may be applied to the CTM in order to solve it using the RLT method. Figure 4.1 shows the relationship between these relaxations.
The CTM with quadratic objective function admits a convex quadratic programming relaxation and a linear programming relaxation. The least absolute deviation (LAD) objective admits a linear programming relaxation. These relaxations are discussed in the next three sections.

4.2.1 A convex quadratic programming relaxation of the quadratic CTM

Recall that the quadratic CTM problem may be written as

\begin{align*}
\text{Minimize } & \quad z = \sum_{i=0}^{n} \sum_{k=1}^{m} w_i^k \left( \hat{T}_i^k - T_i^k \right)^2 \\
\text{s.t. } & \quad \sum_{j=1}^{n} r_{0j}(T_j^k - T_0^k) = -q \quad \forall k = 1, \ldots, m \tag{4.6a} \\
& \quad \sum_{j \in \{0, \ldots, n\} \setminus i} r_{ij}(T_j^k - T_i^k) = h_i^k S_i T_i^k \quad \forall i = 1, \ldots, n, \forall k = 1, \ldots, m. \tag{4.6b}
\end{align*}
This formulation of the CTM contains second order terms in the form of $(T_k^i)^2$ for $i = 0, \ldots, n$, $k = 1, \ldots, m$ in the objective function and, $r_{ij}T_k^i$ and $r_{ij}T_k^j$ for $i = 0, \ldots, n$, $j = i + 1, \ldots, n$, and $k = 1, \ldots, m$ in the constraint set. If we define the variables $U_{ij}^k = r_{ij}T_k^i$ and $V_{ij}^k = r_{ij}T_k^j$, and substitute these variables into the constraints, then we have generated a relaxation consisting of a convex quadratic objective and linear constraints.

$$\text{minimize} \sum_{k=1}^{m} \sum_{i=0}^{n} w_k^i (\hat{T}_k^i - T_k^i)^2$$  \hspace{1cm} (4.7a)

$$\text{s.t.} \sum_{j=1}^{n} U_{0jk} - V_{0jk} = -q \quad \forall k = 1, \ldots, m$$  \hspace{1cm} (4.7b)

$$\sum_{j \in \{0, \ldots, n\} \setminus i} U_{ijk} - V_{ijk} = h_k^i S_i T_k^i \quad \forall k = 1, \ldots, m \quad \forall i = 1, \ldots, n$$  \hspace{1cm} (4.7c)

The constraints (4.8a) through (4.8h) are the bound factor constraints implied on the $U$ and $V$-variables by the upper and lower bounds of the $T$ and $r$-variables. (The indices on these bounds are omitted for clarity.) The RLT algorithm may proceed in this case by branching on the $r$-variables, solving at each step a convex quadratic programming problem relaxation of (4.6).
4.2.2 A linear programming relaxation of the quadratic CTM

If we define the variables $U_{kij} = r_{ij}^T k_j$ and $V_{ij} = r_{ij} T_i^k$, and substitute these variables into the constraints as in the quadratic programing relaxation above, and further define the variables $X_{ki} = (T_i^k)^2$ for $i = 0, \ldots, n$ and $k = 1, \ldots, m$, then we can form a linear programming relaxation of (4.6):

\[
\text{minimize} \quad \sum_{i=0}^{n} \sum_{k=1}^{m} w_{ki} X_{ki} - 2w_{ki}^{kT} T_i^k + \sum_{k=1}^{m} \sum_{i=0}^{n} w_{ki} (T_i^k)^2
\]

\[
\text{s.t.} \quad \sum_{j=1}^{n} U_{0jk} - V_{0jk} = -q \quad \forall k = 1, \ldots, m
\]

\[
\sum_{j \in \{0, \ldots, n\} \setminus i} U_{ijk} - V_{ijk} = h^k S_i^k \quad \forall k = 1, \ldots, m, \forall i = 1, \ldots, n.
\]

The objective (4.9a) includes a constant term, which we keep in the implementation so that direct comparisons between heuristic and RLT solutions may be made. The bound factor constraints associated with the variables $X, U, V$ are given by:

\[
-X_{ki}^k + u_T T_i^k + l_T T_i^k \geq l_T u_T \quad \forall i, k
\]

\[
X_{ki} - 2l_T T_i^k \geq -(l_T)^2 \quad \forall i, k
\]

\[
X_{ki}^k - 2u_T T_i^k \geq -(u_T)^2 \quad \forall i, k
\]

\[
U_{ijk} - l_T r_{ij} \geq -l_T l_T \quad \forall i = 0, \ldots, n, \forall j > i, \forall k = 1, \ldots, m
\]

\[
U_{ijk} - u_T T_i^k - u_T r_{ij} \geq -u_T u_T \quad \forall i = 0, \ldots, n, \forall j > i, \forall k = 1, \ldots, m
\]

\[
-V_{ijk} - u_T T_i^k - u_T r_{ij} \geq -u_T u_T \quad \forall i = 0, \ldots, n, \forall j > i, \forall k = 1, \ldots, m
\]

\[
-V_{ijk} + l_T r_{ij} \geq -l_T l_T \quad \forall i = 0, \ldots, n, \forall j > i, \forall k = 1, \ldots, m
\]

\[
-V_{ijk} + u_T T_i^k + u_T r_{ij} \geq -u_T u_T \quad \forall i = 0, \ldots, n, \forall j > i, \forall k = 1, \ldots, m
\]

\[
-V_{ijk} + l_T r_{ij} \geq -l_T l_T \quad \forall i = 0, \ldots, n, \forall j > i, \forall k = 1, \ldots, m.
\]
Note that one constraint induced by the bound factors on the $X$-variables is redundant and is omitted from this enumeration. In this problem relaxation, it is necessary to branch only on the $T$-variables in order to find a solution that is feasible to the original quadratic CTM.

4.2.3 A linear programming relaxation of the least absolute deviation CTM

The least absolute deviation form of the CTM is:

Minimize $z = \sum_{i=0}^{n} \sum_{k=1}^{m} w_i^k |\hat{T}_i^k - T_i^k|$ \hspace{1cm} (4.11a)

s.t. $\sum_{j=1}^{n} r_{0j}(T_j^k - T_0^k) = -q \hspace{0.5cm} \forall k = 1, \ldots, m$ \hspace{1cm} (4.11b)

$\sum_{j \in \{0, \ldots, n\} \setminus i} r_{ij}(T_j^k - T_i^k) = h_i^k S_i T_i^k$ \hspace{1cm} \forall i = 1, \ldots, n, \forall k = 1, \ldots, m. \hspace{1cm} (4.11c)

The linear programming relaxation of this form of the CTM first linearizes the objective function by introducing variables $E_i^k$ and $E_i^k$ for $i = 0, \ldots, n$ and $k = 1, \ldots, m$. These correspond the positive error and negative error in each term of (4.11a). The nonlinear terms in the constraint set are relaxed as in the other two problem relaxations with the introduction of $U$ and $V$-variables. This results in a linear relaxation that takes the form:

Minimize $\sum_{i=0}^{n} \sum_{k=1}^{m} w_i^k (E_i^k + E_i^k)$ \hspace{1cm} (4.12a)

s.t. $\sum_{j=1}^{n} U_{0jk} - V_{0jk} = -q \hspace{0.5cm} \forall k = 1, \ldots, m$ \hspace{1cm} (4.12b)

$\sum_{j \in \{0, \ldots, n\} \setminus i} U_{ijk} - V_{ijk} = h_i^k S_i T_i^k \hspace{0.5cm} \forall k = 1, \ldots, m \hspace{0.5cm} \forall i = 1, \ldots, n. \hspace{1cm} (4.12c)$
\[ E_{ki}^k \geq \hat{T}_{ki}^k - T_{ki}^k \quad \forall i = 1, \ldots, n, \ \forall k = 1, \ldots, m \quad (4.13a) \]

\[ E_{ki}^k \geq T_{ki}^k - \hat{T}_{ki}^k \quad \forall i = 1, \ldots, n, \ \forall k = 1, \ldots, m \quad (4.13b) \]

\[ E_{ki}^k \geq 0 \quad \forall i = 1, \ldots, n, \ \forall k = 1, \ldots, m \quad (4.13c) \]

\[ E_{ki}^k \geq 0 \quad \forall i = 1, \ldots, n, \ \forall k = 1, \ldots, m \quad (4.13d) \]

\[ -X_{ki}^k + u_T T_{ki}^k + l_T T_{ki}^k \geq l_T u_T \quad \forall i = 1, \ldots, n, \ \forall k = 1, \ldots, m \quad (4.13e) \]

\[ X_{ki}^k - 2l_T T_{ki}^k \geq -(l_T)^2 \quad \forall i = 1, \ldots, n, \ \forall k = 1, \ldots, m \quad (4.13f) \]

\[ X_{ki}^k - 2u_T T_{ki}^k \geq -(u_T)^2 \quad \forall i = 1, \ldots, n, \ \forall k = 1, \ldots, m \quad (4.13g) \]

\[ U_{ijk} - l_r T_{ki}^k - l_T r_{ij} \geq -l_r l_T \quad \forall i = 0, \ldots, n, \ \forall j > i, \ \forall k = 1, \ldots, m \quad (4.13h) \]

\[ U_{ijk} - u_r T_{ki}^k - u_T r_{ij} \geq -u_r u_T \quad \forall i = 0, \ldots, n, \ \forall j > i, \ \forall k = 1, \ldots, m \quad (4.13i) \]

\[ -U_{ijk} + u_r T_{ki}^k + l_T r_{ij} \geq u_r l_T \quad \forall i = 0, \ldots, n, \ \forall j > i, \ \forall k = 1, \ldots, m \quad (4.13j) \]

\[ -U_{ijk} + l_r T_{ki}^k + u_T r_{ij} \geq l_r u_T \quad \forall i = 0, \ldots, n, \ \forall j > i, \ \forall k = 1, \ldots, m \quad (4.13k) \]

\[ V_{ijk} - l_r T_{ki}^k - l_T r_{ij} \geq -l_r l_T \quad \forall i = 0, \ldots, n, \ \forall j > i, \ \forall k = 1, \ldots, m \quad (4.13l) \]

\[ V_{ijk} - u_r T_{ki}^k - u_T r_{ij} \geq -u_r u_T \quad \forall i = 0, \ldots, n, \ \forall j > i, \ \forall k = 1, \ldots, m \quad (4.13m) \]

\[ -V_{ijk} + u_r T_{ki}^k + l_T r_{ij} \geq u_r l_T \quad \forall i = 0, \ldots, n, \ \forall j > i, \ \forall k = 1, \ldots, m \quad (4.13n) \]

\[ -V_{ijk} + l_r T_{ki}^k + u_T r_{ij} \geq l_r u_T \quad \forall i = 0, \ldots, n, \ \forall j > i, \ \forall k = 1, \ldots, m. \quad (4.13o) \]

Since at optimality one or the other of \( E \) and \( \bar{E} \) must be equal to zero, the constraints (4.13c) and (4.13d) are implied by a set of equality constraints,

\[ T_{ki}^k - \bar{E}_{ki}^k + \bar{E}_{ki}^k = \hat{T}_{ki}^k \quad \forall i = 1, \ldots, n, \ \forall k = 1, \ldots, m. \quad (4.14) \]

In this relaxation is it sufficient to branch only on the \( r \)-variables in order to achieve a solution feasible to the original least absolute deviation version of the CTM.
4.3 The RLT algorithm

Regardless of the choice of convex relaxation, the RLT algorithm proceeds by an implicit enumeration scheme wherein an upper and lower bound for the global optimum are maintained as the branch and bound tree is explored. The implementation makes use of an initial upper bound by solving a random restart heuristic that uses the heuristic developed in Chapter 3 as an initial step. At any point where a new upper bound is determined, the $T$-variable bounds are also updated in a manner explained in detail in Chapter 5. When the difference between upper and lower bounds is sufficiently small, the problem returns the best feasible solution found, which is either a global optimum to the nonconvex CTM, or an $\epsilon$-global optimum if the termination criteria is $(UB - LB < \epsilon)$.

While the algorithm is exploring nodes high in the search tree, we employ the heuristic occasionally to seek a tighter upper bound by descending from the current (infeasible) solution to the problem relaxation at the current node. Alternatively, when the heuristic is not called, we calculate the current feasible objective at the current set of $r$-variables. Thus, high in the enumeration tree we preferentially run a heuristic descent, while deeper in the tree, as the bounds tighten and the solutions become nearly feasible we simply calculate the feasible objective from the current solution. These steps are taken with the goal of improving the problem upper bound where it is likely that a better upper bound can be found. The heuristic is run with a probability determined by the node depth $p = e^{-\text{depth}_{\text{node}}}$ where the depth of a given node is the number of arcs in the path from that node to the root node and $h_{\text{scale}}$ is a scale factor. The probability with which we choose to run the heuristic from the current solution decreases as we explore nodes that are deeper in the tree. This is analogous to the cooling schemes used in applications of simulated annealing [36], however, the value of $p$ calculated in our algorithm is not comparable to a Boltzmann probability, $p = e^{-\frac{dE}{kT}}$ employed in the simulated annealing method. While
the Boltzmann probability introduces the possibility of searching in the neighborhood of feasible solutions whose objective values are higher than that of an incumbent solution, our technique for generating probabilities serves a different purpose. Since high in the node tree the solutions to the convex relaxations are not likely to be feasible to the nonlinear problem, we introduce this scheme to search the neighborhood of the infeasible solution. In contrast to the temperature cooling schemes employed in simulated annealing, the argument in the exponential changes only with node depth, rather than with a temperature and difference in objective value.

The following pseudocode description gives an overview of the algorithm.

**Initialization:** Read data. Execute a random restart heuristic using the heuristic algorithm developed in Chapter 3. This step provides an initial feasible solution and an incumbent objective function value \((UB)\). Using this solution update the bounds on the \(T\)-variables as in Chapter 5. Set the initial bounds on the \(r\)-variables. Generate and solve the root relaxation (master problem). Add the root to the node list and set \(LB\) to the the root objective.

**While** \((UB – LB > \epsilon)\)

Select branching node from the top of the node list.

Select a branching variable, \(v\), currently equal to \(v^*\).

Branch on branching variable, \(v\).

Generate left child with \(v \leq v^*\).

Modify master problem with parent cuts + left child cut.

Solve master problem.

**if** (Problem is feasible and objective < \(UB\)).

**if** (Solution is nonlinear-feasible)

Update \(UB\) and \(T\)-bounds.

**else**

Add left child to node list.

endif
endif
if (uniform(0,1) random number < test probability)
    Run heuristic from current r-solution.
    Update UB if a better solution is found.
else
    Determine nonlinear-feasible solution given r.
    Update UB if a better solution is found.
endif
Reset master problem.
Generate right child with $v \geq v^*$. Modify master problem with parent cuts + right child cut.
Solve master problem.
if (Problem is feasible and objective $< UB$).
    if (Solution is nonlinear-feasible)
        Update UB and T-bounds.
    else
        Add left child to node list.
    endif
endif
endif
if (uniform(0,1) random number $< test$ probability)
    Run heuristic from current r-solution.
    Update UB if a better solution is found.
else
    Determine nonlinear-feasible solution given r.
    Update UB if a better solution is found.
endif
Reset master problem
Update LB
End While

Return The optimal solution. \( r, T \).

A flowchart schematic of this procedure is provided in Figure 4.2.

**Figure 4.2.** Flowchart of the RLT procedure.
Note that the algorithm begins with an application of random restart using the heuristic described in Chapter 3. The results of this random restart procedure provide an upper bound and bounds on the $T$-variables, as well as a region in the $r$-variables in which to search for a global optimum using the branch and bound procedure. The solution at the root node provides a lower bound, and the lower bound is always equal to the objective value of the node at the top of the node list. Node list sort order is maintained each time a new node is placed on the node list. This guarantees that when the algorithm selects the top node it is selecting a node whose objective function value is equal to the current lower bound. At any time that a node is solved and found to be tight, a test is performed to determine if a new upper bound has been found. If the tight node provides a new upper bound, then the upper bound and the bounds on the $T$-variables are updated, and the node list is pruned of all nodes whose objective value is greater than the new upper bound.

This algorithm is known to terminate with a globally optimal solution. However in practice the performance of this algorithm implemented naively is inadequate for the practical solution of problems of useful dimensions. Techniques for enhancing the performance of this algorithm are the subject of the next chapter.
Chapter 5

IMPROVING THE PERFORMANCE OF RLT

In this chapter, we examine some techniques for improving the performance of the RLT algorithm described in Chapter 4. As noted in Chapter 1, the performance of algorithms that use an implicit enumeration of problem relaxations may be improved by tightening these relaxations as much as possible. In applying the RLT algorithm to the CTM problem, we face a problem illustrated by the schematic shown in Figure 5.1.

![Figure 5.1. Schematic description of the problem of lower bounds.](image)

The left hand portion of this figure shows a two dimensional representation of an \( r \) hyperrectangle such as might be found at a particular node of the branch and bound tree. This region in the \( r \)-space maps onto some space in the \( T \)-variables shown in the schematic as a representative two dimensional space. The ellipse encloses those solutions in the \( T \)-space that are better than the incumbent upper bound. The center
of the ellipse is the best possible solution, given by $T = \hat{T}$, with an objective value of $z^* = 0$. The problem we face in generating useful lower bounds at each node is illustrated by the schematic mapping of the set of feasible $r$-variables in the rectangle to the nonconvex set of feasible $T$-solutions. If we knew that this set contained no point inside the ellipse, we could fathom this node and rule out the possibility of a better solution in the entire rectangle. Unfortunately, until the $r$ rectangles are made quite small by branching to a significant depth in the branch and bound tree, RLT typically provides only the weak relaxation illustrated by the dotted line in the figure. The relaxation shown includes the point $T = \hat{T}$, which provides no information about a problem lower bound. The node cannot be fathomed, and we must further partition the $r$ hyperrectangle by branching.

This chapter is organized into three sections. In Section 5.1 we describe a set of bounds on linear combinations of the $T$-variables that arise as a consequence of a problem upper bound. These bounds on linear combinations of the $T$-variables may be used to approximate the hyperellipse of $T$-solutions that solve the CTM problem with an objective less than or equal to the objective of the solution that provides our current upper bound. In Section 5.2 we develop an alternative formulation of the problem, replacing the free variables $r$ with nonnegative variables, $\delta$, and the free variables $T$ with nonnegative variables $\Delta$. This permits the introduction of some valid inequalities based on the $T$-bounds developed in Section 5.1. In Section 5.3, we reformulate the problem again, introducing a new variable, $\tau_{ij}^k$, which represents the difference in temperature across nodes $i$ and $j$ at observation $k$. This formulation reduces the number of quadratic terms and permits the introduction in Section 5.4 of a set of constraints that serve to tighten the relaxation sufficiently to permit the solution of problems of reasonable size.
5.1 Bounding the $T$-variables based on a problem upper bound

We first examine bounds on the $T$-variables that follow directly from the upper bound on the objective value provided by a random restart algorithm using the heuristic described in Chapter 3. These bounds provide valid inequalities on the $T$-variables. The idea behind this strategy is shown in Figure 5.2.

Figure 5.2 depicts a two-dimensional space of $T$-variables, $T_i^k$ and $T_j^k$. The center of the ellipse is the point at which $T_i^k = \hat{T}_i^k$ and $T_j^k = \hat{T}_j^k$. The ellipse is the set of all points $(T_i^k, T_j^k)$ at which $w_i^k(T_i^k - \hat{T}_i^k)^2 + w_j^k(T_j^k - \hat{T}_j^k)^2 = z^*$, where $z^*$ is an incumbent upper bound provided by the heuristic. The horizontal and vertical lines tangential to the ellipse are the upper and lower bounds on $T_i^k$ and $T_j^k$ that result from this
upper bound on the objective value.

\[ T^k_i = \left( \hat{T}^k_i - \sqrt{\frac{z^*}{w^k_i}} \right) \]  

(5.1a)

\[ \bar{T}^k_i = \left( \hat{T}^k_i + \sqrt{\frac{z^*}{w^k_i}} \right) \]  

(5.1b)

In a higher dimension problem each of these bounds will provide a pair of hyperplanes for each \( T \)-variable. The upper right and lower left diagonal lines drawn tangential to the ellipse in Figure 5.2 are a consequence of solving the following two problems:

\[
\begin{align*}
\text{maximize} & \quad \sum_{k=1}^{m} \sum_{i=0}^{n} T^k_i \\
\text{s.t.} & \quad \sum_{k=1}^{m} \sum_{i=0}^{n} w^k_i (\hat{T}^k_i - T^k_i)^2 \leq z^* 
\end{align*}
\]

(5.2a)

\[
\begin{align*}
\text{minimize} & \quad \sum_{k=1}^{m} \sum_{i=0}^{n} T^k_i \\
\text{s.t.} & \quad \sum_{k=1}^{m} \sum_{i=0}^{n} w^k_i (\hat{T}^k_i - T^k_i)^2 \leq z^*. 
\end{align*}
\]

(5.3a)

The upper left and lower right cuts are generated by solving similar optimization problems in the difference of the two variables \( T^k_i, T^k_j \). In higher dimensional problems, there are a combinatorial number of such hyperplanes which may be generated by taking combinations of the sums and differences of the \( T \)-variables. Any combination of a subset of the \( T \)-variables may be used to generate a pair of hyperplanes that are tangential to the hyperellipse enclosing solutions whose objective value is less than that of the incumbent. We will make use of this result in the following sections.

### 5.2 A reformulation using nonnegative variables

The formulations developed in Chapter 4 may be used to find a global solution to the CTM problem; however, in practice employing branch and bound on these problem
relaxations results in an algorithm that has impractical solution times for problems of interesting scale. Consequently, we examine problem reformulations that will permit the useful application of RLT. First, because the problem contains quadratic terms it seems intuitive that we would like to consider the impact of multiplying the inequality constraints like those generated by (5.1) by a particular \textit{r}-variable. This idea can be used to produce a valid inequality constraint in terms of one or more of the quadratic terms. The problem is that if the range of the \textit{r}-variable permits a change of sign, then this procedure will result in an arbitrary change in the sense of the inequality of the constraint. To avoid this complication we reformulate the problem in terms of variables whose range is nonnegative. We accomplish this transformation by noting that in the RLT algorithm, the \textit{r}-variables and \textit{T}-variables are always bounded above and below. Taking these bound to be \( r \in (r, \overline{r}) \) and \( T \in (T, \overline{T}) \) this means that \( r_{ij} = r_{ij} + \delta_{ij} \quad \forall i = 0, \ldots, n, \forall j > i \) where \( \delta_{ij} \in (0, \overline{r}_{ij} - r_{ij}) \). Similarly \( T^k_i = T^k_i + \Delta^k_i \) \( \forall i = 0, \ldots, n \) and \( k = 1, \ldots, m \) where \( \Delta^k_i \in (0, T^k_i - T^k_i) \). In terms of these variables, the original quadratically constrained CTM problem is given by:

\[
\begin{align*}
\text{minimize } z &= \sum_{i=0}^{n} \sum_{k=1}^{m} w^k_i \left( \hat{T}^k_i - (T^k_i + \Delta^k_i) \right)^2 \quad (5.4a) \\
\text{s.t. } \sum_{j=1}^{n} (r_{0j} + \delta_{0j})((T^k_j + \Delta^k_j) - (T^k_0 + \Delta^k_0)) &= -q \quad \forall k = 1, \ldots, m \quad (5.4b) \\
\sum_{j \in \{0, \ldots, n\} \setminus i} (r_{ij} + \delta_{ij})((T^k_j + \Delta^k_j) - (T^k_i + \Delta^k_i)) &= h_i^k S_i(T^k_i + \Delta^k_i) \\
&\forall i = 1, \ldots, n, \forall k = 1, \ldots, m. \quad (5.4c)
\end{align*}
\]

We can carry through the multiplication of the terms and obtain a problem with quadratic terms of the form \( \Delta^2 \) and \( \delta\Delta \). Now by substituting the variables \( U \) and \( V \) for \( \delta\Delta \) and \( X \) for \( \Delta^2 \), we can generate the three relaxations developed in Chapter 4 in terms of decision variables that are always nonnegative. The three objective
functions, quadratic, linear, and least absolute deviation, are given by

\[ z = \sum_{i=0}^{n} \sum_{k=1}^{m} w_i^k (\hat{T}_i^k - T_i^k)^2 + 2w_i^k (\hat{T}_i^k - T_i^k)\Delta_i^k + w_i^k (\Delta_i^k)^2, \] (5.5a)

\[ z = \sum_{i=0}^{n} \sum_{k=1}^{m} w_i^k (\hat{T}_i^k - T_i^k)^2 + 2w_i^k (\hat{T}_i^k - T_i^k)\Delta_i^k + w_i^k X_i^k, \text{ and} \] (5.5b)

\[ z = \sum_{i=0}^{n} \sum_{k=1}^{m} w_i^k (\hat{E}_i^k + E_i^k), \] respectively. (5.5c)

The constraints (5.4b) and (5.4c) are the same in all three formulations and may be simplified to the form

\[ \sum_{j=1}^{n} \delta_{0j} (T_j^k - T_0^k) + U_{0j}^k - V_{0j}^k = -q - \sum_{j=1}^{n} r_{0j} (T_j^k - T_0^k) \quad \forall k = 1, \ldots, m \] (5.6a)

\[ \sum_{j \in \{0, \ldots, n\} \setminus i} \delta_{ij} (T_j^k - T_i^k) - h_i^k S_i \Delta_i^k + U_{ij}^k - V_{ij}^k = h_i^k S_i T_i^k - \sum_{j \in \{0, \ldots, n\} \setminus i} r_{ij} (T_j^k - T_i^k) \quad \forall i = 1, \ldots, n, \forall k = 1, \ldots, m. \] (5.6b)

Note that these constraints, in contrast to the constraints in the RLT relaxation in terms of \(r\) and \(T\)-variables, include linear terms in the \(\delta\) and \(\Delta\)-variables as well as additional constant terms. The RLT bound factor constraints on the \(U\) and \(V\)-variables are simplified by the fact that the lower bound in all cases is zero. Dropping the indices for clarity these bound factor constraints for the \(X\), \(U\), and \(V\)-variables
are given as

\[-X + (\mathbf{T} - \mathbf{T}) \Delta \geq 0 \quad (5.7a)\]
\[X \geq 0 \quad (5.7b)\]
\[X - 2(\mathbf{T} - \mathbf{T}) \Delta \geq -(\mathbf{T} - \mathbf{T})^2 \quad (5.7c)\]
\[U \geq 0 \quad (5.7d)\]
\[U - (\bar{r} - r) \Delta - (\bar{\mathbf{T}} - \mathbf{T}) \delta \geq -(\bar{\mathbf{T}} - \mathbf{T})(\bar{r} - r) \quad (5.7e)\]
\[-U + (\bar{r} - r) \Delta \geq 0 \quad (5.7f)\]
\[-U + (\bar{\mathbf{T}} - \mathbf{T}) \delta \geq 0 \quad (5.7g)\]
\[V \geq 0 \quad (5.7h)\]
\[V - (\bar{r} - r) \Delta - (\bar{\mathbf{T}} - \mathbf{T}) \delta \geq -(\bar{\mathbf{T}} - \mathbf{T})(\bar{r} - r) \quad (5.7i)\]
\[-V + (\bar{r} - r) \Delta \geq 0 \quad (5.7j)\]
\[-V + (\bar{\mathbf{T}} - \mathbf{T}) \delta \geq 0. \quad (5.7k)\]

The constraints (5.7b), (5.7d), and (5.7h) are implemented as bounds on the $\delta \times \Delta$ substitution variables, $U$ and $V$. In the least absolute deviation formulation we have the following constraints in addition:

\[\mathbf{E}_i^k \geq \hat{T}_i^k - (\mathbf{T}_i^k + \Delta_i^k) \quad \forall i = 0, \ldots, n, \quad \forall k = 1, \ldots, m \quad (5.8a)\]
\[\mathbf{E}_i^k \geq (\mathbf{T}_i^k + \Delta_i^k) - \hat{T}_i^k \quad \forall i = 0, \ldots, n, \quad \forall k = 1, \ldots, m \quad (5.8b)\]
\[\overline{E}_i^k \geq 0 \quad \forall i = 0, \ldots, n, \quad \forall k = 1, \ldots, m \quad (5.8c)\]
\[\overline{E}_i^k \geq 0 \quad \forall i = 0, \ldots, n, \quad \forall k = 1, \ldots, m. \quad (5.8d)\]

Since one or the other of $\mathbf{E}$ and $\overline{E}$ must be equal to zero, the constraints (5.8a) and (5.8b) may be written as the single constraint set

\[\Delta_i^k - \overline{E}_i^k + E_i^k = \hat{T}_i^k - T_i^k \quad \forall i = 0, \ldots, n, \quad \forall k = 1, \ldots, m. \quad (5.9)\]
5.3 A reformulation using fewer quadratic terms

As a final reformulation of the problem relaxation let us introduce a variable \( \tau_{ij}^k = \Delta^k_j - \Delta^k_i \). Note that since we have lower bounds of zero on all \( \Delta \)-variables, and given a problem upper bound we have an upper bound on these variables as well, the bounds on \( \tau \) are also known: \( \tau_{ij}^k \in (-\Delta^k_i, \Delta^k_j) \). The problem reformulation (in quadratic objective) written in terms of the variables, \( \delta, \Delta, \) and \( \tau \) is provided here:

\[
\begin{align*}
\text{minimize} & \quad z = \sum_{i=0}^n \sum_{k=1}^m w^k_i \left( \hat{T}^k_i - (T^k_i + \Delta^k_i) \right)^2 \\
\text{s.t.} & \quad \sum_{j=1}^n (r_{0j} + \delta_{0j}) (T^k_j - T^k_0 + \tau_{0j}^k) = -q \quad \forall k = 1, \ldots, m \\
& \quad \sum_{j \in \{0, \ldots, n\} \setminus i} (r_{ij} + \delta_{ij}) (T^k_j - T^k_i + \tau_{ij}^k) = h_i^k S_i (T^k_i + \Delta^k_i) \\
& \quad \forall i = 1, \ldots, n, \forall k = 1, \ldots, m \\
& \quad \tau_{ij}^k = \Delta^k_j - \Delta^k_i \quad \forall i < j, \forall k = 1, \ldots, m.
\end{align*}
\]

This formulation results in quadratic terms \( \Delta^2 \) and \( \delta \tau \). The three convex programming problem relaxations of (5.10) then take the form:

\[
\begin{align*}
\text{minimize} & \quad z \\
\text{s.t.} & \quad \sum_{j=1}^n (r_{0j} + \delta_{0j}) (T^k_j - T^k_0 + \tau_{0j}^k) = -q \quad \forall k = 1, \ldots, m \\
& \quad \sum_{j \in \{0, \ldots, n\} \setminus i} (r_{ij} + \delta_{ij}) (T^k_j - T^k_i + \tau_{ij}^k) = h_i^k S_i (T^k_i + \Delta^k_i) - \sum_{j \in \{0, \ldots, n\} \setminus i} \tau_{0j}^k (T^k_j - T^k_i) \\
& \quad \forall i = 1, \ldots, n, \forall k = 1, \ldots, m \\
& \quad \tau_{ij}^k = \Delta^k_j - \Delta^k_i \quad \forall i < j, \forall k = 1, \ldots, m.
\end{align*}
\]

The objectives for the linear, quadratic, and least absolute deviation relaxations remain the same as in (5.5). The bound factor constraints on the \( X \)-variables are as in (5.7a) through (5.7c). The \( U \)-variables, which linearize the terms \( \tau \Delta \), induce bound
factor constraints:

\[ U - \tau \delta - \varpi \delta \geq -\tau \delta \]  \hspace{1cm} (5.12a)
\[ U - \tau \delta \geq 0 \]  \hspace{1cm} (5.12b)
\[ -U + \tau \delta \geq 0 \]  \hspace{1cm} (5.12c)
\[ -U + \delta \tau + \tau \delta \geq \tau \delta. \]  \hspace{1cm} (5.12d)

Note that there is no need to introduce the set of variables \( V \) as in earlier formulations. Further, note that these bound factor constraints may be written in terms of the upper and lower bounds of the \( r \) and \( T \)-variables by the substitution:

\[ \delta_{ij} = r_{ij} - l_{ij} \]  \hspace{1cm} (5.13a)
\[ \tau_{ij}^k = \Delta_{ij}^k = T_{ij}^k - T_{ij} \]  \hspace{1cm} (5.13b)
\[ l_{ij}^k = -\Delta_i^k = -T_i^k + T_i. \]  \hspace{1cm} (5.13c)

The least absolute deviation formulation includes the constraints (5.8).

### 5.4 Valid inequalities that tighten the RLT relaxation

In this section we take the relaxation developed in the previous section and introduce valid inequalities that serve to tighten the relaxation. The first set of these inequalities are generated from the bounds discussed in in Section 5.1. Let us have an upper bound on the problem, \( z \). Then we have for each \( \Delta_i^k \)-variable,

\[ \Delta_i^k \leq -\hat{T}_i^k + T_i^k + \sqrt{\frac{z}{w_i^k}} \]  \hspace{1cm} (5.14a)
\[ \Delta_i^k \geq -\hat{T}_i^k + T_i^k - \sqrt{\frac{z}{w_i^k}} \]  \hspace{1cm} (5.14b)
\[ \Delta_i^k \geq -\hat{T}_i^k + T_i^k - \sqrt{\frac{z}{w_i^k}} \]  \hspace{1cm} (5.14c)
Now take one expression to be in the index \( j \) and the other to be in \( i \) and multiply the inequality in \( i \) by -1 giving:

\[
\Delta^k_j \leq -\hat{T}^k_j + T^k_j + \sqrt{\frac{z}{w^k_j}}
\]

(5.15a)

\[
-\Delta^k_i \leq \hat{T}^k_i - T^k_i + \sqrt{\frac{z}{w^k_i}}
\]

(5.15b)

\[
\Delta^k_j \leq \hat{T}^k_j - T^k_j + \sqrt{\frac{z}{w^k_j}}
\]

(5.15c)

This approach provides two expressions for \( \tau^k_{ij} = \Delta^k_j - \Delta^k_i \):

\[
\tau^k_{ij} = \Delta^k_j - \Delta^k_i \leq -\hat{T}^k_j + T^k_j + \sqrt{\frac{z}{w^k_j}} + \hat{T}^k_i - T^k_i + \sqrt{\frac{z}{w^k_i}}
\]

(5.16a)

\[
\tau^k_{ij} = \Delta^k_j - \Delta^k_i \geq -\hat{T}^k_j + T^k_j - \sqrt{\frac{z}{w^k_j}} + \hat{T}^k_i - T^k_i - \sqrt{\frac{z}{w^k_i}}.
\]

(5.16b)

Figure 5.3 is an illustration of the effect of this \( \tau \) transformation. Given an upper bound on the objective function value we have upper and lower bounds on the variables \( T_i \) and \( T_j \).

Figure 5.3. An illustration of the effect of the \( \tau \) formulation.
The variables, $\Delta_i$ and $\Delta_j$ are bounded below by zero and above by $\overline{T} - \overline{T}$. Naively, the $\tau$ variables are bounded by $\Delta_j$ and $-\Delta_i$ because of the equality $\tau_{ij} = \Delta_j - \Delta_i$. However, note the point marked “X” in Figure 5.3. The point $(\overline{T}_i, \overline{T}_j)$ corresponds to the point at which $\Delta_i = 0$ and $\Delta_j$ is at its maximum value. This should be the maximum allowable value for $\tau$, but it is not a point of interest because it lies outside the hyper-ellipse for which we already have a better incumbent solution. A similar argument holds for the points marked “Y,” $(\overline{T}_i, \overline{T}_j)$. As presented in Section 5.1, the terms involving the upper bound on the objective function value may be scaled by a factor of $\sqrt{2}$. Now multiply each side of the resulting inequalities (5.16) by $\delta_{ij}$. This cannot change the sense of the inequalities because the $\delta$-variables are nonnegative. The resulting constraints are in terms of the linearization variables, $U_{ij}^k$:

\begin{align*}
\delta_{ij} \tau_{ij}^k &= U_{ij}^k \leq \delta_{ij} \left( -\hat{T}_{j}^{k} + T_{j}^{k} + \sqrt{\frac{z}{2w_{j}^{k}}} + \hat{T}_{i}^{k} - T_{i}^{k} + \sqrt{\frac{z}{2w_{i}^{k}}} \right) \quad (5.17a) \\
\delta_{ij} \tau_{ij}^k &= U_{ij}^k \geq \delta_{ij} \left( -\hat{T}_{j}^{k} + T_{j}^{k} - \sqrt{\frac{z}{2w_{j}^{k}}} + \hat{T}_{i}^{k} - T_{i}^{k} - \sqrt{\frac{z}{2w_{i}^{k}}} \right). \quad (5.17b)
\end{align*}

In an analogous manner to that described in Section 5.1 we may take the sum over all observations of the $\tau$-variables and multiply through by a particular $\delta_{ij}$-variable to generate a constraint for each arc $(i, j)$ in the network.

\begin{align*}
\sum_{k=1}^{m} U_{ij}^k &\leq \sum_{k=1}^{m} \delta_{ij} \left( -\hat{T}_{j}^{k} + T_{j}^{k} + \sqrt{\frac{z}{2w_{j}^{k}}} + \hat{T}_{i}^{k} - T_{i}^{k} + \sqrt{\frac{z}{2w_{i}^{k}}} \right) \quad (5.18a) \\
\sum_{k=1}^{m} U_{ij}^k &\geq \sum_{k=1}^{m} \delta_{ij} \left( -\hat{T}_{j}^{k} + T_{j}^{k} - \sqrt{\frac{z}{2w_{j}^{k}}} + \hat{T}_{i}^{k} - T_{i}^{k} - \sqrt{\frac{z}{2w_{i}^{k}}} \right). \quad (5.18b)
\end{align*}

By including the inequalities developed in Section 5.1 written in terms of the $\Delta$-variables, along with those given by (5.17) and (5.18) we are able to sufficiently tighten the RLT relaxation to permit the solution of problems of useful dimension. Experimental results are presented in Chapter 7.
Chapter 6

SOFTWARE DESIGN

The software developed to explore this problem made use of recent practice in software design and was novel enough to warrant some description. The software is written in a combination of C++ and Java™. Further references to Java and related software will omit the explicit trademark sign. A complete description of the Java programming language is available at [79]. The software is built of modular components that communicate through existing application programmer interfaces. The most critical of these are the Java Native Interface (JNI), and the Ilog CPLEX interface. The JNI allows a Java application to access native compiled C++ object files. This permits the software developed in this paper to execute efficient native functions for numerically intensive procedures that require fast execution while still permitting a modular, machine independent, and object oriented design for overall program execution. In Section 6.1 we give an overview of the framework under which the software was designed. In Section 6.2, we examine this framework from a functional perspective. In section 6.3, we provide details on some of the more important components of the software package.

6.1 Software Framework

The overall software architecture may described as having three layers shown in Figure 6.1.
At the top layer is a Java application that controls the execution of the algorithms. This layer contains the implementation of a branch and bound algorithm and the node and node tree data structures required for its execution. This layer also contains the provision for using the heuristic in a random restart algorithm, and for keeping track of the current status of the algorithm and the best incumbent solution. The layer also permits outputting current solution data and data about the node tree and individual active nodes. Memory management, including the management of native object memory allocation is provided through the Java Virtual Machine and the Java System object’s garbage collection function. In order to solve a CTM using this software, the user invokes a Java program.

Figure 6.1. Software framework.
The bottom two layers are actually a monolithic compiled C++ shared object file, `librlt.so`. The shared object file is loaded by the class loader of the Java Virtual Machine at run time using the command `System.loadLibrary(librlt.so)`. Once this natively compiled shared object library is loaded into the JVM, all of the functions and object definitions implemented in it are available to the running Java program. This object is able to run the optimization problems natively through the CPLEX API, which is represented as a third layer in Figure 6.1. CPLEX provides an API to the objects that make up an optimization problem, including variables, constraints, objectives, and solution algorithms. The API permits access to these objects for problem generation, modification, and for the analysis of the current state of variables at a particular point in problem execution. This software makes use only of the CPLEX API and its included simplex and barrier solvers. A complete description of the CPLEX API is provided in [77].

### 6.2 Functional Framework

The software may be described functionally by the following enumeration:

**Data Reading** Reading a standard data file and generating and populating the data structures required to solve the RLT. These functions are provided by the native class `CTMDataReader`. The data structures used in the problem are defined in the file `rltGlobals.h`.

**Heuristic Solver** The object file `CTMHeuristic` is a native C++ object that solves the heuristic problem based on the data in the structures populated by the reader. This object also provides functions for finding a current feasible solution from a fixed set of $r$-variables, and for determining the bounds on the $T$-variables given an upper bound on the objective value.

**rltMaster** The `rltMaster` object contains the RLT problem relaxation and is where
relaxations of the CTM are solved through CPLEX. The class file `rltMaster.java` is an abstract class that is subclassed by concrete `rltLinearMaster`, `rltQuadMaster`, `rltLeastMaster`, and `rltTauMaster` classes whose implementations provide the details of generating, solving, and modifying the relaxations required by RLT. This class hierarchy is shown in Figure 6.2.

![Figure 6.2. Master class diagram.](image)

**Node and Node List Management** The node list, consisting of the set of current active nodes is implemented on the Java side. Node management is accomplished by the class files:

- `RLTNode`
- `RLTNodeList`
- `RLTCut`
- `RLTNodeComparator`

The `RLTNode` class permits the implementation, modification, and solution (through the JNI) of individual nodes. Each node contains a reference to the single master problem. Nodes also contain a list of the bound cuts that are
applicable at that node. Branching on a node results in the generation of two child nodes that contain the cut implied by the current solution status of the designated branching variable. Once a node has been selected and its children generated and solved, it is removed from the active node list and the memory allocated to it is recovered by the garbage collector of the JVM. Similarly, if the solution at a node is found to be infeasible or of greater objective value than the incumbent, it is removed from the node list. This procedure is known as fathoming [17]. The $\text{RLTNode}$ list is backed by a $\text{TreeSet}$ from the Java $\text{Collections}$ interface. This $\text{TreeSet}$ is sorted according to the logic provided by the $\text{RLTNodeComparator}$. The node object uses a Java $\text{Vector}$ structure to store instances of $\text{RltCut}$ objects. These contain an individual bound cut on the node’s cut list. A diagram of this relationship is shown in Figure 6.3.

![Diagram of Node List and Node Structure](image)

**Figure 6.3.** Node list and node structure.
The node comparator is used by the TreeSet to maintain the sort order of the list according to objective value, then node depth, and finally node index. Node indices are guaranteed to be unique by the class level logic of the RltNode object. This guarantee of a unique sort further guarantees node insertions and deletions within the TreeSet. The TreeSet object is a concrete implementation of the AbstractSet object specified in the Collections framework [79]. It has log(n) time performance for the insertion, deletion, and “contains” operations. Since the set is not synchronized, care was taken in the software design to prohibit altering the list while iterating over it. While pruning the node list, nodes are marked for deletion, and then deleted once the iterator has been exhausted.

6.3 Software Components

6.3.1 The RltNode component

The RltNode object contains most of the attributes and methods required for sub-problem management. A detailed diagram of the RltNode object is provided in Figure 6.4.
The add cut logic provides access to a unique list of branching variable bound cuts contained within each node. The branching logic generates left and right children with the appropriate cuts added based on the selection of a branching variable. The nodes are sensitive to their master problem type, and the embedded logic distinguishes between the branching and non branching variables at the interface to the master problem. When the `branch(int)` method is invokd by a running Java program, the node on which the operation was called spawns two child nodes. Each of these children is a copy of the parent node with the addition of a single branching cut on the variable indexed by the integer passed as an argument. The right hand side value of this cut is equal to the current solution status of that variable at the parent node. Once these child problems are generated they are solved in turn. First, the single master problem object is modified. The variable bounds in the master are updated to reflect the bounds in the child problem’s cut list. The node is then solved via a call to

**Figure 6.4.** The RLT node object.
the `solveNode()` method. This method invokes the CPLEX solver on the modified problem which executes the CPLEX LP solver in the case of linear relaxations, and the barrier solver in the case of quadratic programming relaxations. Once the solver terminates, the `solveNode()` method examines the solution first to determine if the solver terminated with a feasible solution. If the problem at a particular node is infeasible, then the node is not added to the node list, and no further action is necessary. If the problem reaches a feasible solution, the solution is examined to determine if it is a tight solution. That is, the solution status of the relaxed variables, $U, V$, and possibly $X$, are compared to the solution status of variables for which they are surrogates: $rT, T^2, \delta\Delta$, or $\delta\tau$. If every substitute variable is within a tightness epsilon of the product of variables for which it is a surrogate, then the problem at this node is declared tight. The solution at this node is compared to the incumbent upper bound solution, and if is a better (lower objective value) solution, then the incumbent is updated. Whenever the incumbent solution is updated, the bounds on the $T$-variables, and some of the constraints mentioned in Chapter 5 may also be updated. In any case tight nodes may be fathomed and thus are not added to the node list. If the solver terminates with a feasible solution, and the solution is not tight, then the node is added to the node list for further exploration. Once execution of the `solveNode()` method is returned to the `branch()` method, the master problem is repaired. That is, the cuts added by the child node being solved are removed and the native CPLEX model object is returned to the root node state. Once a node has been branched, and its children potentially added to the node list, the parent is removed from the node list.

### 6.3.2 The `CTMHeuristic` component

The `CTMHeuristic` object is implemented natively. It provides a heuristic solver and access to the CPLEX variable bounds.
Figure 6.5. The CTM heuristic object.

The logic for updating bounds on the $T$-variables based on a new incumbent upper bound on the problem is contained in the `updateTVariableBounds()` method of the `CTMHeuristic` object. The sorting algorithm used by the heuristic is an adaptation of the quicksort routine [49]. The `CTMHeuristic` object also provides bisection search and custom matrix manipulation functions. The underlying matrix object is defined in `matrix.h`, a free version of the TechSoft Matrix implementation [78]. The heuristic solver is invoked from a starting position. Provisions are made for setting this starting position. An additional function of the `CTMHeuristic` object is the “getCurrentFeasibleSolution()” function. This function takes a current (perhaps infeasible) set of $r$-variables, and simply solves the $m$ systems of linear equations implied by these variables and the problem data to generate the set of corresponding temperatures and the objective function value. This function is used in the overall global optimization scheme as the bounds on the $r$-variables at a particular node become fairly tight. At these nodes it is useful to note the quality of the feasible solution that corresponds to the solution given by the RLT relaxation, as there exists the possibility that this solution will be better (lower objective value) than the current incumbent solution. The generation of random starting positions for the heuristic to use in a random restart approach is implemented on the Java side along with the random
restart algorithm. Calls to the heuristic solver may be made with a default set of $\epsilon$ and iteration parameters, or with a completely specified set of these parameters.

### 6.3.3 The RLTMaster component

The RltMaster object provides an abstract definition of an object that permits the generation, modification, and solution of optimization problems used by the RLT branch and bound tree as well as access to information about the solution. The main solution program, running in a Java Virtual Machine, is able to access the native implementations of the master problem class objects which in turn have access to the CPLEX model objects and solution information through the Ilog CPLEX api.

<table>
<thead>
<tr>
<th>RLTMaster</th>
</tr>
</thead>
<tbody>
<tr>
<td>int Master Problem Type</td>
</tr>
<tr>
<td>Generate Master</td>
</tr>
<tr>
<td>Solve Master</td>
</tr>
<tr>
<td>Alter Problem</td>
</tr>
<tr>
<td>Repair Problem</td>
</tr>
<tr>
<td>Is Master Tight</td>
</tr>
<tr>
<td>Get Branching Variable</td>
</tr>
<tr>
<td>Get Variable and Solution Statuses</td>
</tr>
</tbody>
</table>

**Figure 6.6.** The RLT master problem object.

This abstract Java class is implemented by four concrete classes which correspond to the four solution approaches described in Chapter 4. The `generateMaster()` method takes the information contained in the problem’s (already populated) native data structures, and builds the objective function and constraint arrays that comprise the root RLT relaxation of the problem which is either a linear program, or a convex quadratic program. When the node object invokes the native `solve()` command, the master object creates a CPLEX model from the constraint arrays and objective...
function and invokes the applicable CPLEX solver (either LP or barrier). Once the solver terminates, the solve method inserts solution information into the appropriate data structures. These solution information data structures are accessible to the Java side node object through setter and getter methods which pass values through the JNI. The master problems also include some utility and analysis functionality such as dumping problem information to standard output. Standard output on the C native side is passed to the System object loaded in the JVM and behaves identically to a Java call to `System.out`.

### 6.3.4 The RLTSolver component

The Java class `RltSolver` contains the main method used to solve a CTM problem. The main method of the solver takes as arguments a data file name and the integer constant corresponding to an RLT relaxation type from the enumeration `{ALG_QUADRATIC, ALG_LINEAR, ALG_ABSLEASTDEV, ALG_TAU}`. This enumeration, along with several other problem constants used on the Java side are defined in the class file `rltConstants.java`. The `rltSolver` implements a random restart algorithm, a branch and bound algorithm and a function for determining whether specified stopping criteria have been met. These are invoked by the main method in order to seek a solution to a particular instance of a CTM. A significant advantage of the software design is a consequence of this design decision. The user may, with very little effort, and with the compilation of only a single Java class file, construct any solution approach that makes use of the already compiled components and their embedded functionality. For example, this software may be used strictly as a heuristic. Alternatively, the heuristic may be employed in-line with the branch and bound algorithm. In fact, we run the heuristic occasionally from solutions to the relaxation that are high in the branch and bound tree in the hope of descending to a solution that is better than that of the current incumbent.
6.3.5 The **DataReader** component

The data reader parses a standard input file and populates the native data structures used by the solver. At run time, the solver instantiates a Java object of class *DataReader* and invokes the *readData()* method on this object. In fact the data reader is composed of native C functions. As an example of how the Java Native Interface is used throughout the software presented here consider Figure 6.7.

![Diagram of DataReader component](image)

**Figure 6.7.** Node list and node structure.

This figure illustrates the method of generating a native shared object library that may be accessed by a running Java program through the JNI. The .h file is automatically generated by the command *javah* invoked on a compiled Java class that contains
method definitions, which are declared using the _native_ keyword. The developer implements these definitions in a .cpp source file and compiles them normally. The only additional requirement for this procedure to work is that the implementation .cpp file include the library _jni.h_.

6.3.6 Extending the use of the software

The software discussed in this chapter provides two contributions for further development. First it serves as a concrete example of a system that uses a modern modular, extensible and object oriented design, while using the execution speed of native C++ shared objects and the CPLEX API through the Java Native Interface. This has proven to be a useful framework for solving the CTM problem using RLT, but can be extended to many complex or large scale problems that resort to multiple sub-problem and master problem modifications under an implicit enumeration scheme. Secondly, the software might be more directly adapted to similar problems that use an RLT relaxation scheme.
Chapter 7

COMPUTATIONAL EXPERIENCE

In this chapter we provide the results of a series of computational trials that have been performed on the algorithms laid out in this paper.

The data sets used in these tests are described in the following table:

Table 7.1. CTM data sets.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Data Set Name</th>
<th>Number of Nodes</th>
<th>Number of Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>BGA Four Areas</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>B</td>
<td>BGA Four Areas Split</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>C</td>
<td>BGA Split Bottom</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>D</td>
<td>CPGA Lid Five Areas</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>E</td>
<td>CPGA Lid Four Areas</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>F</td>
<td>CPGA Lid Four Areas Split</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>G</td>
<td>CPGA Partial Socket</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>H</td>
<td>CPGA Partial Socket Split</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>I</td>
<td>Delphi TQFP</td>
<td>5</td>
<td>99</td>
</tr>
<tr>
<td>J</td>
<td>Delphi BGA</td>
<td>5</td>
<td>99</td>
</tr>
<tr>
<td>K</td>
<td>Delphi BGA Split</td>
<td>7</td>
<td>99</td>
</tr>
</tbody>
</table>

The nine node data sets were used by Devoe [21], and the 99 node sets are test cases nominated by Lasance et al. [43]. These sets are characteristic of Plastic Ball Grid Array (BGA), Ceramic Pin Grid Array (CPGA), and Thin Quad Flat Pack (TQFP) chip packages under boundary conditions for heat transfer coefficients ($h$-values) that permute extreme cases at the nodes in order to stress the CTM model being tested. We first examine the use of the heuristic technique in a random restart algorithm. These experimental trials were designed to characterize the performance of the algorithm and to suggest parameter values that provide useful results with reason-
sonable computation effort. Next we explore the effectiveness of the RLT algorithm, of which the heuristic is an important component in generating exact solutions and problem lower bounds to which these heuristic solutions may be compared.

7.1 Heuristic Results

7.1.1 Computational time versus solution quality

We begin by making some general observations regarding the computational behavior of our heuristic. In the larger data sets, there were on the order of 600 critical points to sort using our implementation of the quicksort routine. The separation of critical points into two bins that can be sorted separately provides a substantial savings in the computational cost of the sorts with almost no effort. Using a larger number of bins proved difficult, since boundary values that would place a roughly equal number of critical points into each bin are difficult to ascertain a priori. The implicit enumeration scheme seems to be especially efficient. While the sorted search list contained from 50 to 600 sub-regions, the algorithm typically searched only about five before fathoming the rest of the list by bound.

Next, note that by decreasing our $\epsilon$-values within the heuristic and by permitting more total iterations before the stopping criteria are satisfied, we may hope to improve the quality of the solution obtained by our heuristic. However, these gains come at the cost of incurring additional computational expense, which may not justify a small improvement in objective quality. Thus, we examined the computational time required to execute our heuristic versus the solution quality obtained when certain algorithm parameters were varied as shown in Table 7.2.
Table 7.2. Conditions for comparison.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Condition 1</th>
<th>Condition 2</th>
<th>Condition 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inner optimization $\epsilon_1$</td>
<td>0.0005</td>
<td>0.00001</td>
<td>0.00005</td>
</tr>
<tr>
<td>Inner optimization count</td>
<td>50</td>
<td>300</td>
<td>50</td>
</tr>
<tr>
<td>Bisection $\epsilon_2$</td>
<td>0.0001</td>
<td>0.0001</td>
<td>$1 \times 10^{-8}$</td>
</tr>
<tr>
<td>Bisection count</td>
<td>25</td>
<td>25</td>
<td>100</td>
</tr>
</tbody>
</table>

Condition 1 is a baseline condition that seems to be a reasonable compromise between solution quality and computational time. For condition 2, we allowed the inner optimization step to run to a smaller relative improvement epsilon and increased the corresponding iteration limit count to a larger value. Under condition 3, we keep the inner optimization conditions as in condition 1, but use a much smaller $\epsilon_2$ and correspondingly larger iteration limit in the bisection searches of sub-regions of the objective function. Table 7.3 shows the CPU time and final objective function values generated by running the heuristic on each of the data sets in Table 7.1 from an initial starting position in which $R_{ij} = 0.001, \forall 0 \leq i < j \leq n$, using the different conditions for algorithm parameters shown in Table 7.2.

Table 7.3. Comparison of CPU times and objectives under different conditions.

<table>
<thead>
<tr>
<th>Data Set Name</th>
<th>CPU time (1)</th>
<th>CPU time (2)</th>
<th>CPU time (3)</th>
<th>Objective (1)</th>
<th>Objective (2)</th>
<th>Objective (3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1.11</td>
<td>2.14</td>
<td>1.66</td>
<td>$1.78 \times 10^{-3}$</td>
<td>$1.58 \times 10^{-4}$</td>
<td>$1.72 \times 10^{-4}$</td>
</tr>
<tr>
<td>B</td>
<td>1.55</td>
<td>6.67</td>
<td>2.05</td>
<td>$1.40 \times 10^{-2}$</td>
<td>$1.24 \times 10^{-2}$</td>
<td>$1.40 \times 10^{-2}$</td>
</tr>
<tr>
<td>C</td>
<td>0.22</td>
<td>0.46</td>
<td>0.42</td>
<td>3.12</td>
<td>3.09</td>
<td>3.11</td>
</tr>
<tr>
<td>D</td>
<td>1.06</td>
<td>14.97</td>
<td>3.92</td>
<td>$1.62 \times 10^{-2}$</td>
<td>$1.54 \times 10^{-4}$</td>
<td>$1.03 \times 10^{-3}$</td>
</tr>
<tr>
<td>E</td>
<td>1.37</td>
<td>7.75</td>
<td>1.75</td>
<td>$9.58 \times 10^{-4}$</td>
<td>$3.86 \times 10^{-5}$</td>
<td>$9.47 \times 10^{-4}$</td>
</tr>
<tr>
<td>F</td>
<td>0.74</td>
<td>15.38</td>
<td>0.92</td>
<td>$2.41 \times 10^{-2}$</td>
<td>$1.52 \times 10^{-2}$</td>
<td>$2.44 \times 10^{-2}$</td>
</tr>
<tr>
<td>G</td>
<td>0.64</td>
<td>7.92</td>
<td>1.68</td>
<td>$3.40 \times 10^{-2}$</td>
<td>$1.55 \times 10^{-4}$</td>
<td>$1.73 \times 10^{-2}$</td>
</tr>
<tr>
<td>H</td>
<td>1.01</td>
<td>9.21</td>
<td>0.96</td>
<td>$2.00 \times 10^{-2}$</td>
<td>$1.30 \times 10^{-2}$</td>
<td>$1.48 \times 10^{-1}$</td>
</tr>
<tr>
<td>I</td>
<td>48.67</td>
<td>191.54</td>
<td>53.66</td>
<td>$2.99 \times 10^{-2}$</td>
<td>$2.79 \times 10^{-2}$</td>
<td>$3.00 \times 10^{-2}$</td>
</tr>
<tr>
<td>J</td>
<td>37.86</td>
<td>1320.84</td>
<td>43.43</td>
<td>$1.74 \times 10^{-1}$</td>
<td>$1.63 \times 10^{-1}$</td>
<td>$1.74 \times 10^{-1}$</td>
</tr>
<tr>
<td>K</td>
<td>48.38</td>
<td>189.44</td>
<td>54.52</td>
<td>$2.99 \times 10^{-2}$</td>
<td>$2.79 \times 10^{-2}$</td>
<td>$3.00 \times 10^{-2}$</td>
</tr>
</tbody>
</table>

CPU time (i): CPU seconds required under condition i
Objective (i): objective value obtained under condition i

The results in Table 7.3 show us first that there is a tradeoff between solution
quality and computational time. The difference between condition 1 and condition 2 is most stark. When a tight relative objective improvement $\epsilon_1$ is used, the heuristic performs many more sequences of coordinate descents. These are costly in terms of computation time, but in most cases do not generate dramatically improved objective values. Data set $E$, in which we observe nearly two orders of magnitude of improvement in objective value, is an exception. The results of increasing the precision of the bisection search are also worthy of note. Recall that bisection search (see e.g. [17]) is used to find a local minimum within each sub-region. We found that terminating our bisection search procedure within an interval of uncertainty of 0.0001 consumes reasonable computational resources while returning an adequately precise estimation of a local optimum in the individual sub-region searches. Substantially decreasing the magnitude of the bisection $\epsilon_2$ and increasing the corresponding iteration limit resulted in modest improvement in some objective values at a cost in computation expense that was significant, but not as dramatic as the increases seen with condition 2. Of particular note, in some cases we observe a small increase in the objective value found using the tighter bisection search criteria.

7.1.2 Examining starting position

In this experiment we started the heuristic using each of eight data sets from five different initial solutions. These initial solutions were $R_{ij} = 10^{-k}, \forall 0 \leq i < j \leq n$, for each $k = 1, \ldots, K$. All other parameters were set according to condition 1 as given in Table 7.2. The results of this experiment are given in Table 7.4 and show that the heuristic is quite sensitive to the initial starting position. The best objective value found for each data set is underlined.
Table 7.4. Objective values for different starting positions.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>0.1</th>
<th>0.01</th>
<th>0.001</th>
<th>0.0001</th>
<th>0.00001</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1.26×10^{-1}</td>
<td>2.18×10^{-3}</td>
<td>1.78×10^{-3}</td>
<td>2.86×10^{-1}</td>
<td>3.44×10^{-1}</td>
</tr>
<tr>
<td>B</td>
<td>4.83×10^{-1}</td>
<td>3.27×10^{-2}</td>
<td>1.40×10^{-2}</td>
<td>4.68×10^{-1}</td>
<td>1.31×10^{-2}</td>
</tr>
<tr>
<td>C</td>
<td>2.84</td>
<td>2.97</td>
<td>3.12</td>
<td>3.33</td>
<td>3.43</td>
</tr>
<tr>
<td>D</td>
<td>2.56×10^{-3}</td>
<td>3.11×10^{-4}</td>
<td>1.62×10^{-2}</td>
<td>9.97×10^{-4}</td>
<td>2.40</td>
</tr>
<tr>
<td>E</td>
<td>9.33×10^{-4}</td>
<td>6.90×10^{-4}</td>
<td>9.58×10^{-4}</td>
<td>1.14×10^{-3}</td>
<td>1.17×10^{-3}</td>
</tr>
<tr>
<td>F</td>
<td>2.53×10^{-2}</td>
<td>3.66×10^{-2}</td>
<td>2.41×10^{-2}</td>
<td>3.09×10^{-1}</td>
<td>2.53×10^{-2}</td>
</tr>
<tr>
<td>G</td>
<td>7.30×10^{-4}</td>
<td>2.60×10^{-4}</td>
<td>3.40×10^{-2}</td>
<td>2.66×10^{-2}</td>
<td>2.27×10^{-1}</td>
</tr>
<tr>
<td>H</td>
<td>2.54×10^{-2}</td>
<td>3.42×10^{-2}</td>
<td>2.00×10^{-2}</td>
<td>2.39×10^{-1}</td>
<td>8.02×10^{-2}</td>
</tr>
<tr>
<td>I</td>
<td>6.34×10^{-2}</td>
<td>3.91×10^{-2}</td>
<td>2.99×10^{-2}</td>
<td>6.16×10^{-2}</td>
<td>2.06×10^{-1}</td>
</tr>
<tr>
<td>J</td>
<td>3.46×10^{-1}</td>
<td>1.62×10^{-1}</td>
<td>1.74×10^{-1}</td>
<td>4.86×10^{-1}</td>
<td>4.92×10^{-1}</td>
</tr>
<tr>
<td>K</td>
<td>6.34×10^{-2}</td>
<td>3.91×10^{-2}</td>
<td>2.99×10^{-2}</td>
<td>6.16×10^{-2}</td>
<td>2.06×10^{-1}</td>
</tr>
</tbody>
</table>

In almost every case different starting positions can lead to objectives that differ by orders of magnitude. This is a consequence of the solution space given by (3.1b) and (3.1c) which leads to many sub-optimal local optima. We note that for the data sets we examined, starting positions near 0.01 and 0.001 seem reasonable, but that there might be better starting positions that differ substantially. Further, our fixed starting positions set $R_{ij}$ equal to some value for all $R_{ij}$. A better starting position can almost certainly be found in a place where the $R_{ij}$ do not all take on the same value. This experiment also indicates that it may be worthwhile to explore better methods of finding a good starting solution. In the next section we perform an experiment that examines this suggestion.

### 7.1.3 Random restart

The sensitivity to starting position and the apparent costliness of allowing the algorithm to converge to very small improvement suggested that our heuristic could be used effectively as a step in a random restart algorithm. We tested a random restart algorithm using fifty randomly generated starting positions whose starting $R$-values were independent randomly generated values drawn from a normal distribution with
mean 0.0055 and standard deviation 0.0045. These parameters were chosen because they were centered between the two values that tended to produce the best objectives in the experiment in Section 7.1.2. We bracketed those values at one standard deviation from the center resulting in a search space that included the regions that seemed most promising. The algorithm was run under condition 1 from Section 7.1.1. The random start position values were generated using the standard C random library, *ranlib.c*. The best objectives found at ten-iteration increments are shown in Table 7.5.

**Table 7.5.** Progress of random restart over 50 trials.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Number of Restarts</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>$1.80 \times 10^{-3}$</td>
</tr>
<tr>
<td>B</td>
<td>$1.36 \times 10^{-2}$</td>
</tr>
<tr>
<td>C</td>
<td>3.14</td>
</tr>
<tr>
<td>D</td>
<td>$1.57 \times 10^{-4}$</td>
</tr>
<tr>
<td>E</td>
<td>$1.12 \times 10^{-3}$</td>
</tr>
<tr>
<td>F</td>
<td>$1.44 \times 10^{-1}$</td>
</tr>
<tr>
<td>G</td>
<td>$7.93 \times 10^{-4}$</td>
</tr>
<tr>
<td>H</td>
<td>$3.28 \times 10^{-2}$</td>
</tr>
<tr>
<td>I</td>
<td>$3.19 \times 10^{-2}$</td>
</tr>
<tr>
<td>J</td>
<td>$1.63 \times 10^{-1}$</td>
</tr>
<tr>
<td>K</td>
<td>$3.19 \times 10^{-2}$</td>
</tr>
</tbody>
</table>

As expected, the random restart scheme is able to consistently out-perform a fixed starting position scheme using the same parameters. The random restart scheme appears to make most of its progress by around thirty iterations.
7.2 RLT Results

We tested an implementation of the application of RLT described in Chapters 4 and 5 in order to determine global optima to the data of interest, and to characterize the performance and potential for practical use of the algorithm. Further, these global optima permit an evaluation of the usefulness of the heuristic technique developed in Chapter 3. The results of computational tests on the first eight data sets from Table 7.1 are shown here. These data sets were solved to global optimality using a quadratic objective formulation of the CTM based on the $\tau$ variable substitution described in Chapter 5. A detailed description of the software is provided in Chapter 6. As shown in the flowchart in Figure 4.2 these experiment begin with an application of the random restart heuristic. This starting value is reported in the results along with the best answer found and the system time used to calculate the best answer. The trials were run on a Sun Blade machine running CPLEX 8.1 with 2056MB of installed RAM. The system was free of significant loads other than these computations during the trial. The default settings for the CPLEX barrier optimization solver were used. We performed some experimentation on using other than default values for the $BarEpComp$, $BarItLim$, and $BarAlg$ parameters and found no significant improvement in performance or solution quality. Parameter tuning and the use of different methods for efficiently solving the convex problem relaxations are a promising area for further exploration. Table 7.6 shows results from this experiment.
Table 7.6. Results of an experiment using RLT.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Size</th>
<th>Starting Solution</th>
<th>End Solution</th>
<th>System Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5×9</td>
<td>984.0</td>
<td>187.6</td>
<td>1:37.27</td>
</tr>
<tr>
<td>B</td>
<td>7×9</td>
<td>112622.3</td>
<td>42850.3</td>
<td>39:31.52</td>
</tr>
<tr>
<td>C</td>
<td>6×9</td>
<td>1536.8</td>
<td>4.1</td>
<td>1:04.50</td>
</tr>
<tr>
<td>D</td>
<td>6×9</td>
<td>252853.2</td>
<td>161197.9</td>
<td>8:3231.68</td>
</tr>
<tr>
<td>E</td>
<td>5×9</td>
<td>24904.1</td>
<td>138.4</td>
<td>3:33.47</td>
</tr>
<tr>
<td>F</td>
<td>7×9</td>
<td>120809.4</td>
<td>120809.4</td>
<td>17:40:42.65</td>
</tr>
<tr>
<td>G</td>
<td>5×9</td>
<td>1145.6</td>
<td>4.9</td>
<td>1:02.02</td>
</tr>
<tr>
<td>H</td>
<td>7×9</td>
<td>246515.6</td>
<td>142659.8</td>
<td>37:10.00</td>
</tr>
</tbody>
</table>

This small experiment demonstrates that the RLT is capable of finding better solutions than the heuristic used alone in a random restart algorithm. Solutions to the much larger data sets proved difficult to obtain, and exploring problems whose convex relaxations are large might benefit from the decompositions techniques suggested in Chapter 8.

7.3 An experiment on stability

The Compact Thermal Model provides opportunities for study that go beyond the focus of this work, which is on specific methods for solving the challenging optimization problem embedded within the CTM. As an exploration of one of these topics, and as an indication of a direction for future work, we examined the stability of a solution to one case of the CTM. For this experiment we computed the globally optimal solution to data set “A” described in Table 7.1. We then perturbed the input reference temperature data by a small amount and finally calculated global optima to these perturbed problems. The optima were calculated using a least absolute deviation version of the CTM model. Each reference temperature, \( \hat{T}_i^k \), was perturbed by a random deviation generated from a uniform distribution whose range was ±0.1 percent of the reference value. The results of this experiment are given in Table 7.7.
Table 7.7. Results using randomly perturbed input data.

<table>
<thead>
<tr>
<th>Value</th>
<th>Baseline</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective</td>
<td>80.52</td>
<td>80.93</td>
<td>87.27</td>
<td>78.20</td>
<td>90.74</td>
<td>76.01</td>
</tr>
<tr>
<td>SSD $\Delta \hat{T}$</td>
<td>7.3045</td>
<td>8.5258</td>
<td>6.0185</td>
<td>11.4294</td>
<td>6.6254</td>
<td></td>
</tr>
<tr>
<td>SSD $\Delta T$</td>
<td>4.0593</td>
<td>0.2990</td>
<td>0.5645</td>
<td>0.6167</td>
<td>2.6463</td>
<td></td>
</tr>
<tr>
<td>SSD $\Delta r$</td>
<td>11.2429</td>
<td>1.6472</td>
<td>0.7217</td>
<td>0.4450</td>
<td>3.5218</td>
<td></td>
</tr>
<tr>
<td>$\Delta$ Obj (%)</td>
<td>0.5%</td>
<td>8.4%</td>
<td>-2.9%</td>
<td>12.7%</td>
<td>-5.6%</td>
<td></td>
</tr>
</tbody>
</table>

For each of ten trials the sum of squared differences between the reference and perturbed data sets are presented. These may be compared to the sum of squared differences in the solution $T$ and $r$ vectors and the percent change in the objective value. From this brief experiment it seems apparent that significant changes in the objective value of the CTM problem may be induced by relatively small changes in the input temperatures. The errors induced in this experiment simulate measurement errors of around 1/10 to 1/100 of a degree Celsius. This line of experimentation points to a direction of research into open questions in the CTM methodology that is worthy of further study. Several of these areas are discussed in Chapter 8.
Chapter 8

CONCLUSIONS & RECOMMENDATIONS

In this chapter we draw several conclusions from the work developed and the results we have been able to generate. We also make recommendations on the direction of future research into this particular problem, and to the extension of this work to similar problems.

8.1 Conclusions

The techniques presented here may be applied directly to the various formulations of the CTM problem with little modification. The software not only provides a tool whereby CTM instances may be solved, but also permits the study and extension of the CTM methodology. Such issues as scaling and conditioning, boundary condition sets, and package geometry are all exposed to exploration by the direct application or adaptation of the software. The sensitivity of CTM solutions to small changes in input data may also be explored using the algorithms and software provided by this work.

8.2 Recommendations

In this section, we make two recommendations for extending the research presented. Both of these extensions hold out some promise of improving the performance of the RLT algorithm as applied to this problem, and equivalently of increasing the scale of problems which may be practically solved. The experiments on stability described in Chapter 7 point to an area in which the potential for significant contributions to the overall CTM methodology remains. The performance of the software could be enhanced by parameter tuning and the rigorous application of performance tuning
techniques at the code level. This software is by no means of commercial quality, and any attempt to adapt it to use in an engineering application should include a careful study of techniques to improve its performance and to validate its numerical stability under a wider range of test cases. The sensitivity of solutions to errors in reference temperatures should be thoroughly explored. Additionally, the choice of boundary conditions under which to solve for a descriptive set of $r$-variables is an area in which further research is likely to contribute to the CTM methodology. Both of these areas lie outside the scope of this work, but the existence of this work makes their exploration more practical.

8.2.1 Decomposition techniques

A significant potential extension to this work is suggested by Benders [18]. As the scale of the problems grows, the sub-problems become very large. This is a direct consequence of the linearization procedure, which adds several constraints and additional variable bounds for each linearized term. These constraints are seldom active at the solution to the relaxed problems, and this fact suggests the use of techniques that are well known in their application to large scale linear systems. Since the problem is decomposable, by choice of a feasible set of $r$-variables in to $K$ systems of linear equations, it would seem likely that a decomposition technique might be worth exploring. Unfortunately, the direct application of Benders decomposition to the polynomial problem is not likely to prove possible because of the nonconvex feasible region for $r$. 
However consider a general problem related to the CTM.

\[
\text{Minimize } cx + \sum_{k=1}^{K} (d^k y^k + [xy^k]^T Q^k [xy^k]) \tag{8.1a}
\]
\[\text{s.t. } Ax = b \tag{8.1b}\]
\[
W_i x + T_i^k y^k + [xy^k]^T V_i^k [xy^k] = r_i^k
\]
\[\forall i = 0, \ldots, n, \quad \forall k = 1, \ldots, K \tag{8.1c}\]
\[l_x \leq x \leq u_x, \quad l_{y^k} \leq y^k \leq u_{y^k} \quad \forall k = 1, \ldots, K. \tag{8.1d}\]

If we make an RLT relaxation of the quadratic terms in this problem with a set of variables, \(w\), we generate the linear programming problem

\[
\text{minimize } cx + \sum_{k=1}^{K} (d^k y^k + f^k w^k) \tag{8.2a}
\]
\[\text{s.t. } Ax = b \tag{8.2b}\]
\[
W_i x + T_i^k y^k + G_i^k w^k = s_i^k
\]
\[\forall i = 0, \ldots, n, \quad \forall k = 1, \ldots, K \tag{8.2c}\]
\[
\text{bound factor constraints on } w\text{-variables} \tag{8.2d}\]
\[l_x \leq x \leq u_x, \quad l_{y^k} \leq y^k \leq u_{y^k} \quad \forall k = 1, \ldots, K. \tag{8.2e}\]

At each node of the branch and bound tree we will solve such a relaxation, and the structure of this relaxation admits the application of Benders decomposition in the following way. Let the master problem be given by

\[
\text{minimize } cx + \sum_{k=1}^{K} \Theta^k \tag{8.3a}
\]
\[\text{s. t. } Ax = b \tag{8.3b}\]
\[\Theta^k \geq [\text{a set of Benders cuts}] \tag{8.3c}\]
\[l_x \leq x \leq u_x, \quad \Theta^k \geq 0 \quad \forall k = 1, \ldots, K. \tag{8.3d}\]
The sub-problems may be generated only in terms of a particular $k$, given the values for $\hat{x}$ as:

Minimize $d^k y^k + \mathcal{F}^k w^k$ \hspace{1cm} (8.4a)

s. t. $T_i^k y^k + \mathcal{G}_i^k w_i^k = r_i^k - W_i^k \hat{x}$

$\forall i = 0, \ldots, n$ \hspace{1cm} (8.4b)

[bound factor constraints on $w$ given $\hat{x}$ in terms of $y^k$] \hspace{1cm} (8.4c)

$l_{y^k} \leq y^k \leq u_{y^k}$. \hspace{1cm} (8.4d)

In this procedure cuts generated for a problem in the space $l_x \leq x \leq u_x$, $l_{y^k} \leq y^k \leq u_{y^k}$ will be valid in any subspace. This means that nodes can pass valid cuts to their children which may have the effect of improving the solution speed at each node. A scheme for discarding cuts that have become worthless would need to be incorporated into an algorithm that used this procedure.

**8.2.2 Improving lower bounds**

The performance of a branch and bound algorithm can be improved by finding higher quality lower bounds at each node. There may yet exist such a procedure that exploits the special structure of this problem in a way that provides strong lower bounds at each node. Because of the undefined regions over a general $r$-hyperrectangle, the RLT relaxation must necessarily strongly underestimate the objective function in the region. Efforts to exploit the ideas presented in Chapter 5, perhaps under a linear problem formulation, might generate these tighter lower bounds and are the most promising path toward improving performance.
REFERENCES


