ABSTRACT

HONNAVARA, VINAY S. Cost Optimization by Method of Allocating Software Component Units to Electronic Control Units for Model-Driven Designs. (Under the direction of Professor Paul D. Franzon).

Technological advancement in the semiconductor industry and applications has caused a tremendous growth in vehicular electronics, one of the major hurdles due to this explosion is the increase in wiring beyond proportions. Due to the increase in wiring, mileage, stability and reliability of the automotive is affected. At this critical juncture, model based designs allows a software implementation of the automotive and hence multiple architectures can be designed and evaluated at a faster rate without the necessity of building an actual automotive.

A binary quadratic programming model (BQP) is designed to optimize the wiring and also consider the total cost of the electronics, so that an optimal minimal cost solution can be achieved with respect to the electronics of the automotive. The solution of BQP model will result in the assignment of software components (software tasks) to electronic control units (processors) based on various constraints like memory, performance e.t.c. and to achieve the wiring harness. A software package which will setup the environment, solve the binary quadratic programming model to optimality and provide solution sets will be presented here.

The thesis work will provide benchmarks for various problems and a variant of an actual automobile. These benchmarks range from simple examples to futuristic vehicles and provide an insight into the optimization achieved from a system perspective. A simple example is used to illustrate the parameters of the BQP model and results are presented.

Cost Optimization by Method of Allocating Software Component Units to Electronic Control Units for Model-Driven Designs

by
Vinay Srinivasan Honnavara

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fullfillment of the
requirements for the Degree of
Master of Science

Electrical engineering

Raleigh, North Carolina

2008

APPROVED BY:

Dr. Mo-Yuen Chow	Dr. Thomas. W. Reiland

Dr. Paul. D. Franzon
Chair of Advisory Committee

DEDICATION

To mom, dad, pranav and gautham for their love and support.

BIOGRAPHY

Vinay Honnavara was born on 21^{st} March 1985 in Hyderabad India. He received his Bachelor of Engineering (B.E) degree in Electronics and Communications Engineering from Poojya Dodappa Appa college of Engineering (PDA), Vishveshwaraiah Technological University in 2006. In Fall 2006 he began his graduate studies in the Electrical and Computer Engineering department at North Carolina university, Raleigh, NC. Since Fall 2007 he has been working on a software planning tool for wiring harness under the guidance of Prof. Paul Franzon.

ACKNOWLEDGMENTS

First of all i would like to thank Dr Paul Franzon and Dr Kartik Sivaramakrishnan, for allowing me to use their material (White paper - Software component allocation to electronic control units). Much of their work (problem definition, BQP model, branch and cut algorithm, future work ideas) has been used here, and my work was mainly involved in the implementation of the model and generating benchmarks, this thesis would not have been presented here without their work. I would sincerely like to thank my advisor Dr Paul Franzon for his support and guidance in this project, and for giving me a great opportunity to be a part of his research group. He has always been a source of inspiration and I could not have asked for more from him. I would also wish to thank Dr Thomas Reiland and Dr Mo-Yuen Chow for agreeing to become a part of my committee.

I would like to thank Ravi Jenkal, Prem Swaroop, Sunil Basavarajaiah for their help, tips and advice on handling some of the document related issues. Special thanks to John Wilhelmson for providing internal details on ECUs and sensors for the HONDA CIVIC automobile and also a special mention to Sahar Karimi for her work in the project.

It gives me great pleasure and happiness to have such great friends at NC state, Chetan (for being a constant source of irritation), Subbu (for his lack of taste in anything tasty), Harsha (for providing unlimited entertainment), Rachana (for the group studies and free food), Tanu (for her guitar, dance, violin, yoga practice which was always fun to watch), Sowmya (for making unbelievable deserts), Kiran (for taking me safely to various places-taxi driver), Priyanka (for her "so sweet" shrieks), Ranjith (for many things), Sanath (for movies and cricket), Prem (as an elder brother i can look up to), Divya (for her wit), Kav (for awesome food), Vindi (for beer parties), Sunil (for his advice on life), Kamal (for southpark, heroes and AL Pacino movies), Kishore (for his iphone), Murali (for coffee talks and me being one of the few listeners of his un-understandable jokes), Kiran G (for things best untold), Arjun (for his questions), Ryan (for his language), Pi (for his value), Ravi (for his sarcasm and advice on academia), Mithun (for his perspective on everything from hairpin to aero-plane), Anu (for lunches during career fair), Mei (for his truthfulness and constant blabbers), Suraj (for newbie roomie), Raghav (for the extra butter on food)and Nandan (for being the best guinea pig in the world) will always be a indelible part of my life.

A big hug to Pushpa for being a continual source of inspiration and enthusiasm, for the infinite delicious meals and for so many things that has no bounds.

Last but never the least, I would like to thank my family, Saroja, Seenu, Dingu and Gautham for their love, support and care. It is their TLC and the constant remembrance which motivates me everyday.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Electronics has become one of the most growing factors in present day's automotives. Ease of operability, entertainment,telematics, flexibility, re-programmable features, troubleshooting and automated control of performing the mechanical tasks of an automotive, apart from these it removes the dependency on manual control which reduces problems due to safety, environment and traffic [4] , such widespread usage has contributed to the outburst of the electronics growth. The cost of electronics has now reached a stage where it amounts to about half of the cost of the vehicle for high-end automotives. Figure 1.1 shows the evolution of electronics from the point where electronics was first used in an automotive to present day automotives. Although boasting of its infinite advantages it presents a huge problem as electronic content in automotives increases by every production cycle, the wiring also increases. Technology advances in the semiconductor and the networking area are addressing the wiring problem but network architectures are not standardized and are complex in nature, thus wiring is a good approach to have a working solution for a given design. However with the growth of electronic content in automotives, maintaining a lot of wiring is a burden, with the weight being as high as 100 pounds [3]. One of the major setbacks is that the increase in electronics has always been considered as a hardware problem and thus the testing and verification is also done using the old-age approach of using bench-driven testing. As the paradigm shift occurs towards the model based designs, where instead of an on-board testing, a virtual prototype is done with finer details of electronics inside an automotive, more options can be tried and tested without compromising on time-to-market and the cost of the automotive.

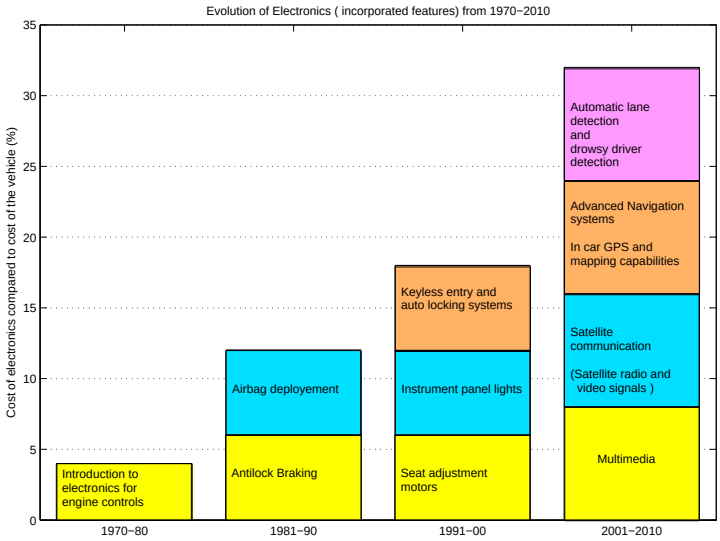


Figure 1.1: Evolution of electronics.

1.1 Motivation

Electronics are presented by building blocks called ECUs (electronic control units) which are board design hosting many processors to perform varied tasks from trivial such as mirror adjustments to complex ones such as the powertrain control. The cost of the ECUs directly affects the cost of the automotive, thus high-end automotives are known to have as many as 100 ECUs. The proliferation is mainly due to entertainment (multimedia) and telematics (radio, GPS). Also along with these, information from sensors, user inputs and housekeeping to name a few. These ECUs must be networked together causing huge wiring. With more and more features and advancements in automotive industry like automatic lane detection, drowsy driver detection [6] automotive companies strive to address the wiring problem. The most simple and popular solution to reduce wiring is to place the ECUs close to the function they intend to perform, for example, brake adjustments control unit is kept as close as possible to the brakes, airbag deployment unit close to the airbags, powertrain control unit close to the engine and so on and so forth. This will cause a proliferation in the number of ECUs which is again unwanted. Thus an overall view of the ECUs and the wiring needs to be considered to actually optimize the architecture. Figures 1.2 and

1.3 indicates the growth of electronic content in automotives for the past three decades, although the electronics growth has been tremendous, the weight due to on board circuitry is quite insignificant when compared to the weight of the automotive which is not the case for wiring, this is mainly due to the enormous technology changes in the semiconductor industry. However the metallurgy industry has not undergone such a huge change which can reduce the weight of the wiring drastically. Although use of aluminum alloys [7] has reduced the mass to an extent, a radical change is necessary to have minimal wiring and achieving the functionality without compromising on safety and reliability.

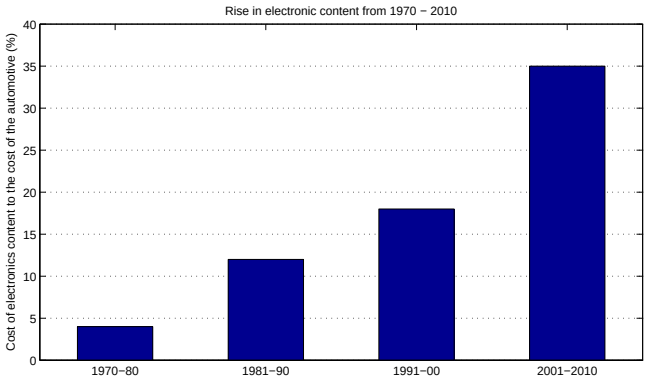


Figure 1.2: Cost of electronics.

1.1.1 AUTOSAR and reconfigurable middleware

With the advancements in reconfigurable middleware, which allows features, functions and services to be added on to the vehicle at a later time, even at post production level, spurs an increased growth in software. As this can be just seen as updating, reconfiguring, upgrading or formatting the software like a 'plug and play' capability, enough overhead is required to visualize these parameters for smooth operation. Also with AUTOSAR (AUtomotive Open System ARchitecture), a consortium by major automotive production companies, trying to set a single standard, so that there could be a competitive market for OEMs (Original Equipment Manufacturer) and reducing the redundancy for the same functionality from different automotive companies, with the basic software and the interfaces being the same for all automotives. The competition among OEMs will cause a rapid

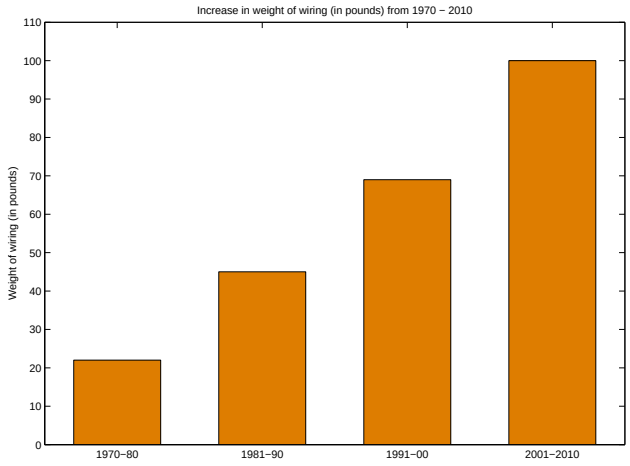


Figure 1.3: Weight of wiring.

growth in embedded electronics again overlooking the wiring problem.

1.1.2 Model-driven design

The old method of testing and verification of an automotive using bench or "plant models" is expensive, cumbersome and causes a slow time-to-market [5]. In a nut-shell, bench based design uses an on-board design and embedded software with connecting wires to check if the desired functionality and all the features are performed, thus actual board designs are needed to test the functions from complex functionality as powertrain management to simple ones like keyless entry. One major limitation is that it causes lot of time and money to actually build the hardware and at the same time presenting maximum performance and reliability. Another limitation is that software engineers have very little understanding on the workings and inner details of the ECUs when troubleshooting is concerned. On the other hand model based design or virtual prototypes offer excellent troubleshooting, where software engineers build the models for the electronics of the automotives. Virtual prototypes provides faster time-to-market with fewer bugs and it uses software programming for mathematical models (VHDL-AMS, SystemC, C) and has powerful analysis tools for performance, reliability, safety and other important parameters. It also

offers a cycle-by-cycle analysis using waveform viewers which will enhance error detection and correction. As stated earlier that automotive companies have started visualizing that the growth in vehicular electronics needs to be provided with a software solution rather than a hardware one, software tool based companies like Mentor Graphics are teaming up with automotive companies to provide solution for such problems (CAPITAL HARNESS^{TM}), such tools offer complete information on ECUs, functions and communication, many architectures can be evaluated in a lesser time than plant models. Thus using grey box testing (or even white box) multiple architectures can be evaluated at a software level, which is faster, and the most optimum one can be considered. Virtual prototypes will soon be embraced by other automotive companies as they realize the limitations of the bench designs.

1.2 Goals

When we consider the bigger picture of increased software content and electronic growth on one hand and on the other, virtual prototype or model-driven design solution, which offers better troubleshooting, is inexpensive and provides options for other architectures to be considered. We would like to propose a model which will optimize the cost of the wiring and in doing so will optimize the entire system since it considers the ECU types and its cost. As stated earlier that one of the most common and popular solutions to keep the ECUs as close to the car parts they service, if this is indeed an optimal solution our model will definitely find it, in case it is not then our model has the potential to return an optimal solution.

We prototype a solution and provide benchmarks for various features of vehicular electronics, ranging from trivial to complicated and also provide a solution for a variant of an actual automobile. We will provide a precise statement for the software component allocation to electronic control units which will setup the architecture and also provide a software which will automatically setup the environment by considering the required parameters, variables, constraints and the objective function and will solve it to optimality.

1.3 Thesis organization

The rest of the documentation will outline on the following aspects. Chapter 2 will give a brief description on the background of vehicle electronics and will review the notation followed in the course of the thesis. Chapter 3 will provide a precise statement on how to allocate software components to electronic control units and will further proceed onto a simple illustrative example on how the parameters will be setup. Chapter 4 illustrates the binary quadratic programming model for achieving the desired cost optimization. Chapter 5 deals with the branch and cut algorithm in solving integer programming models, the algorithm is explained in detail. Chapter 6 deals with the software implementation and provides benchmarks on a range of problems from simplistic problems to complex ones and also deals with the testing and validation involved in this project. Finally we conclude with chapter 7 which offers some discussion and provides information on the future work involved and some practical issues.

Chapter 2

Background

2.1 Overview

In this section, a brief introduction is given about the basic blocks of vehicular electronics and how efficient wiring is done for automotive design. Typical vehicular electronics comprises of 4 basic types, they are,

- **Electronic control units (ECU's):** These are the building blocks of automotive electronics. They comprise of chips on a board protected by a housing. They host a number of processors to perform different tasks and functions, alternatively they are responsible for the technology that drives the car today. It is no surprising factor that some of the high-end cars may have as much as 40-80 ECUs. Typical ECUs are the engine control unit (also known as the ECU), airbag control unit, transmission control unit, speed control unit e.t.c

- **Sensors:** In actual sense sensor is a device which converts a physical quantity into another particular form compatible with an entity that requires it. The entity can be anything , an instrument, another device or an observer. Sensors in automotives provide information about the surroundings to the ECUs, some of them might be for safety or for luxury, some sensors work with actuators for the mechanical motion of the vehicle. The information interpreted by the ECU is processed and in turn performs subsequent operation. A simple example is a collision detection sensor which sends information to the ECU and which will cause the airbag deployment, the chain of

events must happen in real-time.

- **Software component units (SCW's):** These constitute the tasks or the functions which needs to be executed by the ECU, the execution of which causes an operation to be performed. In present day automotives, reconfigurable middleware has caused an enormous software growth for vehicles, thus some of the SCWs also represent the features and services which may not present in present time but can be easily incorporated at a later time.

- **Buses:** Buses represent the channels for interprocess communication. They are available in various types based on their bandwidth, cost and other features.

These units must function accurately and process in real-time. The ECUs and the sensors are wired or networked, since wiring is one of the most tedious tasks as it is both heavy and uses lot of space, the cabling must be done in a manner so that the weight is distributed proportionally to the vehicle. Each year automobile companies design new models having more electronic subsystems has caused an increase in cabling and their characteristics. Most of the wiring harness till 2001 has been done manually, but as automotive companies realize the flaws like complexity, QoS, reliability and labor, wiring design tools have increased, these tools analyze such complex systems and provide a more accurate solution, again with virtual prototyping this comes as a major savior to automotive companies.

2.2 Mathematical Programming models

In this section we will briefly review some of the mathematical programming concepts relevant to the thesis. Mathematical programming is group of study relating to operations research which involves maximizing or minimizing a functional called the objective function, subjected to various constraints. The variables of the function can be real, integer or binary and can have bounds, thus any mathematical programming model is represented by an objective function, constraints and variables. The equation below shows a typical

mathematical programming model

$$\begin{array}{lrcl} \min & (or\max) & f(x) & \\ (s.t.) & Ax & 's' & b \\ & x & \in & \{l,u\} \end{array}$$

Here $f(x)$ represents the objective function and constraints are represented by Ax on the L.H.S side and b of the R.H.S side. The variable x can have lower bound l and upper bound u. The above representation is used to represent any mathematical programming model. Classification of mathematical programming is based on variations of either the objective function (linear, quadratic, non-linear), constraints (linear,quadratic, non-linear) or the variables (real, integer, binary). If there exists a scenario where some variables can take only integer values and others take continuous values then such problems are called mixed integer problems and if they have quadratic coefficients in the objective function then they are called mixed integer quadratic model or MIQP problems, if they can have quadratic coefficients in the constraints then they are called as mixed integer quadratic constrained problems or MIQCP and so on.

Variables (x) that satisfy the constraints represent the feasible solutions. A feasible solution is an optimal solution if no other feasible solution could minimize (or maximize) the objective function. Solutions which do not satisfy the constraints are called infeasible solutions, and solutions where no definite bounds can be determined are called unbounded solutions. Detailed information on mathematical programming is given in [8]. Since our model is a Binary Quadratic programming model (binary variables and quadratic coefficients in the objective function), an understanding of such a model becomes necessary. A general short hand notation for a QP is described below

$$\begin{array}{lrcl} \min & \frac{1}{2}x^TQx & + & c^Tx \\ (s.t.) & Ax & 's' & b \\ & x & \in & \{l,u\}^n \end{array} \tag{2.1}$$

where $Q \in S^n$ represent the quadratic coefficients in the objective function , $c \in \mathbb{R}^n$ represents the linear coefficients in the objective function of n dimensional real vectors, $A \in \mathbb{R}^{m\times n}$ represent the L.H.S of the coefficients of $m \times n$ real matrices, $b \in \mathbb{R}^{p\times n}$ represents the R.H.S of the coefficients, s represents the sense which can be either $=$, $\leq$, or $\geq$ for each of the constraints and finally x represents the variables with bounds l and u.

It is not possible to directly solve a binary model (integer models) using the traditional simplex methods, interior point methods or other algorithms are employed to solve such a model. A continuous QP model is solved in the same way as a LP model with some variations in the simplex method, however QP models usually take lot of time to solve since most of the time is involved in verifying whether the solution is actually the optimal one because unlike linear programming models which consist of only one maxima (or minima) QP models may have numerous maxima (or minima) and to obtain the global maxima (or minima) for the function is extremely tough and in some cases even impossible.

2.3 Review of notation

We briefly review some of the notation we adopt in this document [1]. Let $\mathbb{R}^n$, $\mathbb{R}^{n\times n}$, $\mathcal{S}^n$, denote the set of n dimensional real vectors, $n \times n$ real matrices, and real symmetric matrices of size n, respectively. For a vector $x \in \mathbb{R}^n$, x_i denotes the ith component of this vector. Also, for a matrix $A \in \mathbb{R}^{m\times n}$, a_{ij} will denote the entry in the ith row and jth column of the matrix A. For a set S, $|S|$ denotes the cardinality, i.e, the number of elements of the set. For two sets A and B, $A \subset B$ indicates that A is the subset of B. The notation $x \in \{0, 1\}$ indicates that x is a binary variable. For a non-integer variable x, frac(x) denotes the fractional part of x, i.e., the distance of x from the closest integer. For example, frac$(3.6) = 0.4$. Moreover for a non-integer variable x, $\lfloor x \rfloor$ and $\lceil x \rceil$ denotes the floor and ceiling of x, respectively. For example, $\lfloor 3.1 \rfloor = 3$ and $\lceil 3.1 \rceil = 4$. Also $\overline{x}$ will denote an upperbound value on x and $\underline{x}$ will denote a lowerbound on x.

Chapter 3

Problem setup

We provide a precise problem statement [1] for a system design comprising of ECUs, tasks, sensors and the wiring (or communication) between them, the problem statement is simplified but offers comprehensive information on how the architecture will be designed. Additionally more features and restrictions can be placed on our problem statement to mimic any given example. One such example is provided in this chapter for illustrative purposes which will aid in setting up the architecture.

3.1 Problem statement

We are given a connected bi-directional graph $G = (V, E)$, where the vertex set $V = \{1, 2, \ldots, m\}$ represents m pre-determined locations in the automobile. Each edge represents a direct connection between the two vertices. We now describe the problem statement for the ECU, SCW, sensors and the connections (buses and wires)

1. **ECU (electronic control unit):** An ECU comes in a number of types: $R = \{1, 2, \ldots, p\}$. We will assume that an ECU of type j costs EC_j\$ and has peak performance and memory capacities of EP_j MIPS (millions of instructions per second) and EM_j MB(megabytes), respectively. The cost of the ECU will be a crucial factor for wiring harness; greater the cost will imply greater performance and memory for the ECU. Moreover, an ECU can be placed in any of the m vertices of the graph. We will assume that only one ECU can be placed at each of the vertex of the graph. This assumption simplifies our the model described in chapter 4, and in turn will help

us solve larger instances of the problem. However, we want to emphasize that our models can easily incorporate multiple ECUs and ECU types at a given vertex, at an increase in the model size and with a few modifications.

2. **SCW (Software component unit):** We are given a set $P = \{1, 2, \ldots, n\}$ of software components (SCWs) that perform certain functions in an automobile. The SCWs are to be allocated to ECUs such that each SCW runs entirely on an ECU. An SCW j requires a peak performance of P_j MIPS and M_j MB of memory for its execution. Some of the SCWs communicate with each other to perform some function in the automobile. Let $A = (a_{ij}) \in \mathcal{S}^n$ be the symmetric SCW-SCW adjacency matrix, where $a_{ij} = a_{ji} = 1$ if SCWs i and j communicate with each other, and $a_{ij} = 0$ otherwise. The diagonal entries of A are all zero. We also should have additional regulatory and/or safety requirements that
(a) certain subsets $Q_i, i = 1, \ldots, qn$ of the SCW set P should be included on the same ECU and
(b) certain subsets $R_j, j = 1, \ldots, rn$ of P should NOT be placed on the same ECU. These requirements may be based on the type of the vehicle and may also depend on the safety/stability standards.

3. **Sensors:** Sensors are placed in the first s vertices of the graph. We will assume that there is exactly one sensor at each vertex and vertices can also accommodate an ECU and a sensor. This assumption simplifies our optimization model in (4). We want to emphasize again that our model can also incorporate multiple sensors at a given vertex with slight modifications in the model, we can also implement multiple sensors and ECUs at the vertex. Let $S = \{1, 2, \ldots, s\}$ denote the vertex set for the sensors. Moreover, let $B = (b_{ij}) \in \mathbb{R}^{s \times n}$ be the sensor-SCW adjacency matrix with $b_{ij} = 1$ if sensor i communicates with SCW j and 0 otherwise. There may be additional restrictions that require that
(a) certain sensor-SCW pairs should coexist at given vertex of a graph and
(b) certain sensor-SCW pairs should NOT be located at a given vertex of a graph. Although, we do not consider these constraints in our model in chapter 4, they can be easily incorporated if needed.

4. **Connection and communication:** The edges of the graph G represents the buses

(communication between software component units) and analog wires (connection wires for sensors) which will run along the edges of the graph. Some pairs of SCWs are logically connected together. The logical connection is realized by passing packets through buses that physically connect the pair of ECUs on which these SCWs reside. Moreover, the buses can only run along the edges of the graph G. The buses come in a number of types, whose average cost per unit length per unit bandwidth is \$L. Each pair of communicating SCWs is assigned a bus type. Let us suppose that SCW i located on an ECU at vertex k is logically connected with SCW j located on an ECU at vertex l via a bus of bandwidth BW_{ij} that runs between vertices k and l. Let $\mathcal{P}(k,l)$ be the shortest path between vertices k and l. In this case, one incurs a SCW-SCW connection cost $PP_{kl}^{ij} = \sum_{e \in \mathcal{P}(k,l)} L_{ij} D_e$, where $L_{ij} = BW_{ij} \times L$ and D_e is the length of the edge e in the path. Using this information we construct the SCW i - SCW j connection cost matrix $PP^{ij} \in \mathbb{R}^{m \times m}$. The diagonal entries of the PP^{ij} are all zero i.e., if the SCWs i and j are assigned to the same ECU, there is no logical connection cost.

A sensor is logically wired to a SCW via analog wires that connect the sensor and the ECU on which the SCW resides. These analog wires can only run only along the edges of the graph. Let us suppose that sensor i at vertex i is logically wired to an SCW j located on an ECU at vertex k. Let $\mathcal{P}(i,k)$ be the shortest path between vertices i and k. In this case, one incurs a sensor-SCW connection cost $SP_{ik}^{ij} = \sum_{e \in \mathcal{P}(i,k)} w D_e$, where w is the cost per unit length of an analog wire and D_e is the length of edge e. Using this information, we construct the sensor i - SCW j connection cost matrix $SP^{ij} \in \mathbb{R}^{1 \times m}$. This approach to the cost calculation neglects the potential savings of routing two buses in the same channel. However, we expect the solution that accounts for this savings would be a variant of the form presented here and could be solved as a second step.

The aim is to find a subset of the locations at which to employ the ECUs and then to assign a subset of the SCWs to the employed ECUs so as to minimize the total cost. The total cost is the sum of the costs of the employed ECUs, the total SCW-SCW logical connection costs, and the total sensor-SCW connection costs. Moreover, our assignment must satisfy several constraints that define the set of feasible solutions to the problem. Our objective

is to find an optimal solution that minimizes the total cost. One important thing to note here is that the edges represent both the buses (for SCW communication) and the analog wires (for sensor connection), also the terms connections and communications can be used interchangeably as we are only interested in the wiring harness.

3.2 An illustrative example

We illustrate the problem set up with a simple example in this section [1]. Let us design a simple wiring harness system by connecting 2 sensors to ECUs that might be put in 6 different positions as shown in figure 3.1. Consider the graph $G = (V, E)$, the vertices $V = \{1, 2, 3, 4, 5, 6\}$ of the graph list the locations where the ECU and sensors can be located, and the list of edges $E = \{(1,3), (1,6), (2,5), (3,4), (3,6), (4,5), (4,6), (5,6)\}$ describe permitted wiring channels between the vertex pairs in V. Five SCWs have to be allocated to (yet to be determined) number of ECUs, these SCWs are $P = \{1, 2, 3, 4, 5\}$ and their peak performance and memory requirements are $100, 100, 300, 50, 200$ MIPS and $75, 100, 250, 50, 300$ MB respectively. The SCW-SCW adjacency matrix is

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

The nonzero entries A_{ij} in A correspond to the SCW pairs $i-j$ that communicate with each other and the zero entries represent that the software components do not communicate. For simplicity, we will assume that one ECU type of cost 20$ is available. The performance and memory capacities of this ECU are 500 MIPS and 600 MB, respectively. Moreover, there are no inclusion or exclusion rules in this example, i.e., Q and R are empty sets.

There are two sensors at nodes 1 and 2 in the graph. These sensors communicate with SCWs 1 and 5, respectively. This gives the sensor-SCW adjacency matrix

$$B = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Buses that facilitate the communication between SCWs and also sensors-SCWs are to be assigned. The cost of the bus depends on where each SCW is located. The allocation of SCWs to the vertices of the graph is an outcome of the algorithm. Thus the PP matrices cover the costs of all possibilities. For example, the communication costs between SCWs 1 and 2 depend on where they are located. The following distance matrix

$$D = \begin{pmatrix} 0 & 3 & 1 & 2 & 2 & 1 \\ 3 & 0 & 3 & 2 & 1 & 2 \\ 1 & 3 & 0 & 1 & 2 & 1 \\ 2 & 2 & 1 & 0 & 1 & 1 \\ 2 & 1 & 2 & 1 & 0 & 1 \\ 1 & 2 & 1 & 1 & 1 & 0 \end{pmatrix}.$$

is calculated by running Dijkstra's shortest path algorithm on the graph G and it represents the shortest distance between all the vertex pairs in the graph. Assuming that the cost of the bus type between SCWs 1 and 2 is \$ per meter, we obtain the SCW 1 - SCW 2 wiring cost matrix

$$PP^{12} = \begin{pmatrix} 0 & 6 & 2 & 4 & 4 & 2 \\ 6 & 0 & 6 & 4 & 2 & 4 \\ 2 & 6 & 0 & 2 & 4 & 2 \\ 4 & 4 & 2 & 0 & 2 & 2 \\ 4 & 2 & 4 & 2 & 0 & 2 \\ 2 & 4 & 2 & 2 & 2 & 0 \end{pmatrix}.$$

We can interpret the entries in PP^{12} as follows: For instance, $PP^{12}_{13} = 2$. This tells us that the cost for wiring SCW 1 assigned to an ECU located at vertex 1 with SCW 2 assigned to another ECU located at vertex 3 is 2\$. We can calculate SCW-SCW wiring cost matrices for other pairs of communicating SCWs in a similar fashion.

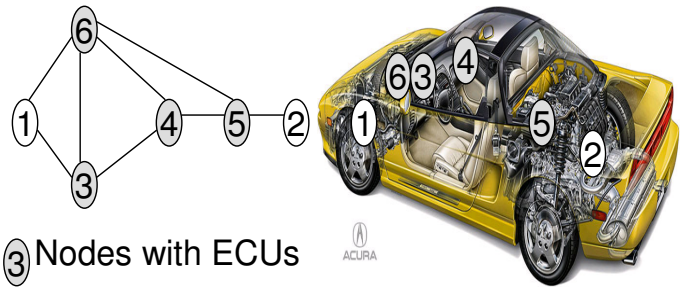
Note: This is one area of the solution that requires heuristic modifiers, since the SCW-SCW wiring cost matrices do not account for sharing of buses between SCWs. Two steps must be taken to anticipate possible sharing. First a-priori, these costs must be reduced by the average amount of sharing expected. If no reduction factor is used, then the algorithm will cost buses too highly compared with the ECUs, and thus will arrive at

the solution requiring too many ECUs. Thus actual sharing must be determined and local optimization and re-allocation is needed.

A similar matrix has to be constructed for the possible sensor-SCW wire runs. It is assumed that no sharing is possible here, since the wires are likely to be analog in nature. Suppose we assume that the cost of the analog wire is 1$ per meter, using this information, we obtain the sensor 1 - SCW 1 wiring cost matrix as

$$SP^{11} = \begin{pmatrix} 0 & 3 & 1 & 2 & 2 & 1 \end{pmatrix}.$$

We can interpret the entries in SP^{11} as follows: For instance, $SP^{11}_{13} = 1$. This tells us that the cost of wiring sensor 1 located at vertex 1 with SCW 1 assigned to ECU at vertex 3 is 1$. We can calculate the sensor-SCW wiring cost matrices for other sensor-SCW communicating pairs in a similar fashion. The connected bi-directional graph G for an automobile is shown in the figure 3.1 (shown automobile in figure for illustrative purposes only) [1], nodes 1 and 2 can accommodate sensors (as well as ECUs) and the other nodes can only accommodate ECUs. The memory and peak performance for SCWs and ECUs along with the cost of an ECU and its type is also shown in the figure. The connection cost matrices for interprocess communication (SCWs) and SCW-sensor communication as PP and SP for all the communicating SCWs and sensor-SCW is illustrated in the figure (the cost communication matrices are formulated based on the bus cost, cost of analog wires and shortest distance using Dijkstra's algorithm).



3 Nodes with ECUs

1 Sensor nodes (can also contain ECUs)

Graph of possible locations and channels
Vertices: V = (1, 2, 3, 4, 5, 6)
Edges: E = edges as drawn

Software Components (SCWs)
P=(1,2,3,4,5)
Performance = (100, 100, 300, 50, 200) (peak MIPS)
Memory = (200, 100, 500, 100, 200) MB.

Communicating SCWs
1-2, 3-4, 1-4, 3-5 (required Bandwidth in Mbps)

$$A = \begin{pmatrix} 0&1&0&1&0 \\ 1&0&0&0&0 \\ 0&0&0&1&1 \\ 1&0&1&0&0 \\ 0&0&1&0&0 \end{pmatrix}$$

ECU types (One type here)
R= (1)
Cost EC1=$20
CPU Performance
EP1 = 500 MIPS
EM1 = 1000 MB
No SCW inclusion or exclusion rules
Q = Φ; R = Φ ;

Sensors
At Vertices: S = (1, 2)
Sensor-SCW adjacency matrix (e.g. sensor 1 communicates with SCW 1):

$$B = \begin{pmatrix} 1&0&0&0&0 \\ 0&0&0&0&1 \end{pmatrix}$$

Buses
SCW-SCW
2 Bus types
SCWs 1,2 use type 1 with cost per meter of $2,
Other pairs use type 2, with cost per meter of $3.

Possible SCW-SCW wiring costs:
SCW 1 to SCW 2

$$PP^{12} = \begin{pmatrix} 0&6&2&4&4&2 \\ 6&0&6&4&2&4 \\ 2&6&0&2&4&2 \\ 4&4&2&0&2&2 \\ 4&2&4&2&0&2 \end{pmatrix}$$

SCW 1 to SCW 4

$$PP^{14} = \begin{pmatrix} 0&9&3&6&6&3 \\ 9&0&9&6&3&6 \\ 3&9&0&3&6&3 \\ 6&6&3&0&3&3 \\ 6&3&6&3&0&3 \\ 3&6&3&3&3&0 \end{pmatrix}$$

$PP^{14} = PP^{34} = PP^{35}$

Sensor –SCW wiring cost
Analog wire with cost per meter of 1$.

Sensor 1 to SCW 1

$SP^{11} = (0\ 3\ 1\ 2\ 2\ 1)$

Sensor 2 to SCW 5

$SP^{25} = (3\ 0\ 3\ 2\ 1\ 2)$

Figure 3.1: An illustrative example.

Chapter 4

A binary quadratic programming model

A binary quadratic programming model is provided here for the problem, the given model is NP-hard (quoting from [9] *"A problem is NP-hard if an algorithm for solving it can be translated into one for solving any NP-problem (nondeterministic polynomial time) problem. NP-hard therefore means "at least as hard as any NP-problem," although it might, in fact, be harder"*). A binary quadratic programming model is one which has quadratic coefficient terms in the objective function and has binary variables. Let $y_i^r = 1$ if an ECU of type r is located at vertex i, and 0 otherwise [1]. Moreover, let $x_{ij} = 1$ if SCW i is assigned to an ECU of some type at vertex j, and 0 otherwise.

We develop the following model for our problem [1]

$$\min \quad \sum_{i=1}^{n}\sum_{j=1}^{m} x_{ij}\left(\sum_{k=1}^{n}\sum_{l=1}^{m} d_{ijkl}x_{kl}\right) \quad + \quad \sum_{r=1}^{p}\sum_{i=1}^{m} EC_r y_i^r \quad + \quad \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} \tag{4.1}$$

$$
\begin{aligned}
\text{s.t.} \quad \sum_{i=1}^{n} P_i x_{ij} - \sum_{r=1}^{p} EP_r y_j^r &\leq 0, \;\; j = 1, \ldots, m \\
\sum_{i=1}^{n} M_i x_{ij} - \sum_{r=1}^{p} EM_r y_j^r &\leq 0, \;\; j = 1, \ldots, m \\
x_{Qk1} j - x_{Qkl} j &= 0 \;\; j = 1, \ldots, m, \;\; k = 1, \ldots, qn, \;\; l = 2, \ldots, |Q_k| \\
\sum_{i \in R_k} x_{ij} &\leq 1, \;\; j = 1, \ldots, m, \;\; k = 1, \ldots, rn \\
\sum_{j=1}^{m} x_{ij} &= 1. \;\; i = 1, \ldots, n \\
\sum_{r=1}^{p} y_i^r &\leq 1, \;\; i = 1, \ldots, m \\
x_{ij} &\in \{0, 1\}, \;\; i = 1, \ldots, n, \;\; j = 1, \ldots, m \\
y_i^r &\in \{0, 1\}, \;\; i = 1, \ldots, m, \;\; r = 1, \ldots, p
\end{aligned}
$$

where,

$$
\begin{aligned}
d_{ijkl} &= a_{ik} PP_{jl}^{ik} \\
c_{ij} &= (B^T SP)_{ij} \\
PP_{jl}^{ik} &= \sum_{e \in \mathcal{P}(j,l)} L_{ik} D_e
\end{aligned}
$$

is the SCW-SCW communication cost and $SP_{ik}^{ij} = \sum_{e \in \mathcal{P}(i,k)} wD_e$ is the sensor-SCW communication cost (refer to problem statement in chapter 3). a_{ik} is the element in the ith row and jth column of the SCW-SCW communication matrix A and B is the sensor-SCW communication matrix (refer to the illustrative example in chapter 3).The explanation for the formalization is given below.

Objective function

- The 1st term in the objective function of (4.1) represents the total SCW-SCW logical connection cost. This is a quadratic term in the objective function because of communication between software units, x_{ij} is the SCW i at vertex j and x_{kl} represents the SCW k at vertex l and cost is represented by d_{ijkl}, thus this communication is a quadratic term.

- The 2nd term in the objective function is the total cost of the ECUs, since ECUs are available in various types and costs, (assuming that higher cost offers better performance and memory capabilities) minimizing the cost of the ECU while considering the harness is extremely important as the trade-off lies in a wise choice between the number of ECUs and the wiring required.

- The 3rd term in the objective is the sensor-SCW connection cost matrix, c_{ij} represents the cost and communication of the sensors with the SCWs and x_{ij} represents the SCW i at vertex j. c_{ij} is of the order $n \times m$ as $c_{ij} = (B^T SP)_{ij}$ where B is of the order $s \times n$ and SP is of the order $s \times m$.

Constraints

- The 1st constraint ensures that the MIPS (performance) requirement is satisfied, since a SCW is assigned to ECU of type r at vertex j, the performance of the SCW should not ensure that of the ECU, thus if multiple ECU types are present more software units can be assigned to these ECUs in a way that the objective function is also minimized.

- The 2nd constraint is similar to the 1st one but in terms of memory, the memory of the SCW should not exceed that of the ECU, thus this constraint needs to be satisfied. Again with multiple ECUs more software units can be assigned considering the minimization of the objective function

- The 3rd set of constraints are for the inclusion requirements. This can be better illustrated with the following example Lets say that SCWs 2,3 and 4 should be included on the same ECU for safety/regulatory requirements. This means that x_{2j}, x_{3j} and x_{4j} should all be 1 or should be 0, this means to say that if an ECU is located at a particular node then software units 2, 3 and 4 should either assigned to ECU or to a different ECU, this is done by making $x_{2j} - x_{3j} = 0$, $x_{2j} - x4j = 0$ and $x_{3j} - x4j = 0$. Thus the inclusion requirement is achieved by the generalized form given in the above equation (4.1). The elements are arranged in ascending order, thus Q_{kl} represents the lth element in set Q_k.

- The 4th constraint represents the various exclusion requirements, this again formulated similar to the inclusion requirement done above but is explained here for illus-

trative purposes. Lets say that SCWs 5 and 6 should not be included on the same ECU, so if x_{5j} and x_{6j} corresponding to SCWs 5 and 6 assigned to ECU at location j respectively. This means that either of them can be 1, but both of them cannot be 1, thus $x_{5j} + x_{6j} \leq 1$ corresponding to the exclusion constraint (4th constraint) in eqn 4.1.

- The 5th constraint ensure that only one SCW type can be assigned to an ECU, this also signifies that one particular SCW cannot be present in multiple ECUs, and all the SCWs are to be assigned to ECUs. For example if the model gives a result of only one ECU, all the software units should be assigned to that ECU.

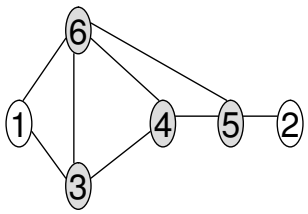
- The 6th constraint ensures that no more that one ECU can be placed at any location, it is possible to incorporate more ECUs at any given location by changing the R.H.S side of this constraint, here it is 1 for simplicity.

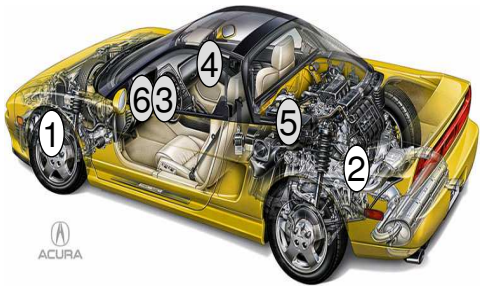
- The 7th and 8th constraints impose binary restrictions on the variables x_{ij} and y_i^r, alternatively the variables x_{ij} and y_i^r can have a lower bound of 0 and an upper bound of 1.

Our model (4.1) is a binary quadratic program (QP) that is NP-hard. We will develop a branch and cut algorithm for solving (4.1) to optimality in section (5.2).

Figure 4.1 [1] shows the optimization variables, at any given node, ECUs and its interface with other nodes (other ECUs) is modeled. As different types of ECUs exist, based on their performance and memory capabilities, $y_{ir} = 1$ if an ECU of type r exist at node i. The other optimization variable is the software component where each of them perform a different function, they are listed based on their performance and memory usage and their communication with other SCWs, inclusion and exclusion requirements of the SCWs should also be accounted. Figure 4.2 shows the optimization model which on solving yields a minimal cost architecture to optimize the wiring (and also the system), as shown in the figure the solution of such a model will yield the assignment of software components to electronic control units, the number of electronic control units is also a part of the solution. The cost communication matrices are formulated by considering the actual costs of buses, gateways, connectors and wiring turns (bus routing phase). Bandwidth allocation for the SCW-SCW communication (assumed a linear relationship here for adding additional bandwidth)

is sufficient enough for the communication to occur, the constraints for the optimization model (figure 4.2) are mainly the memory and the performance criteria of the SCW-ECU assignment (other miscelleneous constraints like inclusion and exclusion requirements, SCW assignment e.t.c are explained in section 4.1)

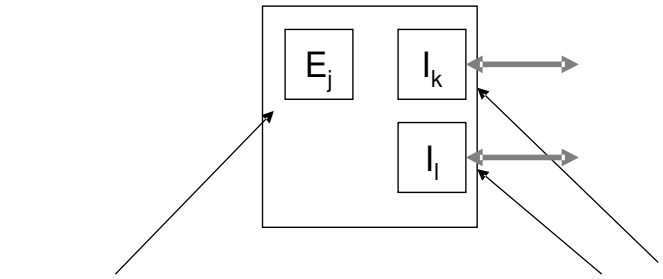




3 Nodes permitting ECUs

1 Sensor nodes (can also contain ECUs)

At each node permitting an ECU:



ECU from a specified list with cost ECj, performance EPj and memory capacity EMj

Interfaces Ik (cost and capacity folded into buses at routing)

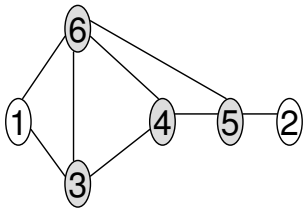
y_{ir}=1 if ECU type r allocated to node i

Software Components (SWC)

- List {SWCj} with performance requirements SPj, memory requirements SMj, and inter-SWC communications requirements SCij
- Constraints: Inclusion (must be on node i); Exclusion (can not be on same node

x_{ij}=1 if SWC i allocated to node j

Figure 4.1: Optimization variables.



Wiring Harness:
SWC/ECU assignment phase: Assume average cost per unit bandwidth and per unit distance : L

Optimization: SWC/ECE Assignment

Allocated Processors to nodes, ECUs to nodes and buses to edges so as to:

Minimize cost $\Sigma\, y_{ir}\, EC_r + \Sigma$ wiring costs

Subject to the following constraints:
- Σ SPj – Σ ECj ≤ 0 (enough performance at each node)
- Σ SMj – Σ EMj ≤ 0 (enough memory at each node)

Bus Routing Phase:

Minimize total harness cost, by treating it as a routing problem, or formulating as an optimization step. Accounts for:
- Actual costs of buses, connectors, turns and gateways
- Providing sufficient SWC-SWC bandwidth through available routing

Figure 4.2: Optimization model.

Chapter 5

Branch and cut scheme

Integer programming problems are solved using various algorithms. There are exact algorithms (provides an optimal solution but known to be inefficient and provides a bound on optimality: branch and bound, cutting planes e.t.c) , approximation algorithms (provides a suboptimal solution with a bound on optimality) and heuristics (provides a suboptimal solution without a bound on optimality: local search, simulated annealing). Although we have used CPLEX optimization engine to solve our model, understanding the algorithm to solve 4.1 is essential. It is emphasized that CPLEX uses branch and cut method largely to solve integer level programs but also applies heuristics to the nodes, albeit it allows the user to set the options on solution type (feasibility, optimality) , memory usage and algorithms. However the model here is incomplete without the algorithm used to solve it and thus will be explained herein.

5.1 Background

Branch and cut is a powerful algorithm to solve integer programming models. It is an implicit enumeration scheme i.e. it intelligently sorts out the feasible solutions to provide an optimal solution. It is a combination of branch and bound scheme and cutting planes algorithm, both these methods are also frequently employed to solve integer or mixed integer programming problems. Branch and bound is very effective and the solution is reliable but consumes lot of computational time to solve integer programming problems to optimality, on the other hand cutting planes algorithms are unstable but takes very less time and

memory to solve problems to optimality. Thus branch and cut algorithm uses both the above schemes and provides a faster and a stable optimum solution. A brief review is given on branch and bound and cutting planes scheme and is followed by the branch and cut algorithm in detail.

5.1.1 Branch and bound algorithm

Typical Linear or non-linear programming problems can be solved using the simplex methods, for integer level problems simplex methods will sometimes yield a fractional result for the variables and thus for the optimum solution value. Thus schemes like branch and bound are used in conjunction with simplex methods to provide integer solutions. It must also be noted that rounding of variables obtained from a relaxed simplex method will not always yield satisfactory results and will be far from the optimal solution. We now proceed with the branch and bound scheme.

To solve an integer linear programming problem (ILP) we solve it by relaxing the integer variables using the simplex method (without the integer restrictions) and obtain the optimal solution

$$z = cx$$

this provides an upperbound solution since the method yields the most optimum solution with the optimized variables (which may not be integers), thus the integer optimum solution will be a part of the solution set of integer solutions. Then from the list of non-integer variable set we pick the one with the most fractional part, $\hat{x_j}$ has the most fractional part if

$$\hat{x_j} \quad = \quad \max[|(x(i) - \lfloor x(i) \rfloor)|] \text{ for i = 1,2,. . . ,n} \tag{5.1}$$

For example if

$$x = (0.25, 1.35, 2.75, 2.45)$$

Then we would pick 2.45 because the fractional part is the highest (0.45 is farthest from the integer), another way of looking at it is 0.45 is closest to $\frac{1}{2}$. Now we branch on it and create two more subproblems from this by adding two constraints, one as $x \leq 2$ and $x \geq 3$. In

general create two subproblems S^1 and S^2 from the feasible solution set S of the relaxation as

$$S^1 = S \cap \{x \in \mathbb{R}^n : x_j \leq \lfloor \hat{x_j} \rfloor\}$$

and

$$S^2 = S \cap \{x \in \mathbb{R}^n : x_j \geq \lceil \hat{x_j} \rceil\}$$

respectively. Now each of the branches are solved and checked for optimality, initially the bounds are set to $z = -\infty$ (for maximization problems) and then each of the nodes are either branched or pruned (explained below in detail). It is also noted that the notation below is for maximization problems and the same will hold good for minimization problems with change in notations.

- **Branching:** If x is not an integer then we branch on the variable which has the most fractional part (shown in equation 5.1).
- **Pruning by optimality/bounds:** If z be the current bound and z^* be the new bound obtained from a node. If $z > z^*$ then the current bound is better and thus the new branch is pruned by bounds (as the new bound is of no use), however if $z < z^*$ then the new bound is better than the current bound and thus the branch is pruned by optimality.
- **Pruning by infeasibility:** If the node selected, results in an infeasible problem (violation of constraints) then the branch is pruned by infeasibility, the reason for this particular type of the occurrence is as we branch to new nodes, we are consistently adding constraints and the new constraint might nullify the constraints formulated in the original problem.

The above process is repeated till all the nodes are pruned either by optimality, bounds or infeasibility, when all the nodes are exhausted the current bound also known as the current incumbent solution will be the most optimal solution. The branch and bound algorithm can be better understood with an example.

5.1.2 Example for branch and bound scheme

Consider the following example to explain the branch and bound scheme [2]

$$\begin{array}{rrrrl} \max z & = & 4x_1 - x_2 & & \\ & s.t. & 7x_1 - 2x_2 & \leq & 14 \\ & & 0x_1 + x_2 & \leq & 3 \\ & & 2x_1 - 2x_2 & \leq & 3 \\ & & & & x_i \in Z, i = 1, 2 \end{array} \tag{5.2}$$

Using simplex method to solve this example yielded the following results $\overline{z} = \frac{59}{7}$ and $x = (\frac{20}{7}, 3)$. Setting the lowerbound on z as $\underline{z} = -\infty$. Considering the two values x_1 is fractional, so we will branch on this variable and create two sub problems as shown in the figure 5.1, thus,

$$S_1 = S \cap \{x : x_1 \leq \lfloor \tfrac{20}{7} \rfloor\} = S \cap \{x : x_1 \leq 2\}$$

$$S_2 = S \cap \{x : x_1 \geq \lceil \tfrac{20}{7} \rceil\} = S \cap \{x : x_1 \geq 3\}$$

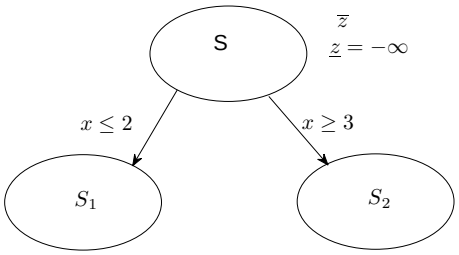


Figure 5.1: Create two child subproblems from parent node.

Let us choose the node S_2 first, and we add the constraint $x_1 \geq 3$ to it i.e.

$$S_1 = S \cap \{x : x_1 \geq 3\}$$

When we observe this node closely we see that the constraints are conflicting, if $x_1 \geq 3$ then the first constraint for the problem $7x_1 - 2x_2 \leq 14$ translates to $21 - 2x_2 \leq 14$ or

$x_2 \geq \frac{7}{2}$ which violates the second constraint of the problem ($x_2 \leq 3$). Thus S_2 is pruned by infeasibility.

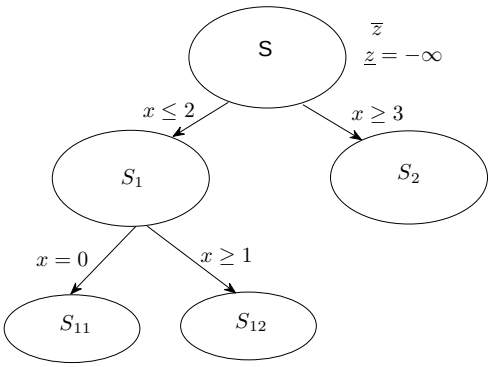


Figure 5.2: Branch and bound (create two more subproblems).

Now let us observe the node S_1 and the constraint $x_1 \leq 2$ to it. Solving it by the simplex method yields the following solution $x = (2, \frac{1}{2})$. Since the result is an integer form we again branch on this variable calling it x_2 and creating two more subproblems from S_1 as shown in the figure 5.2, thus

$$S_{11} = S \cap \{x : x_2 = 0\}$$

$$S_{12} = S \cap \{x : x_2 \geq 1\}$$

Let us consider the node S_{12} (with the constraint $x_2 \geq 1$). The simplex method yields the solution $x = (2, 1)$ which is an optimal solution as $z = -\infty$ and $z^* > z$ thus the bound changes to $z = z*$ or $z = 7$ and conclude this node by stating that node S_{12} is pruned by optimality. One more node needs to be pruned and we will explore the node S_{11} (with the constraint $x_2 = 0$, then the simplex method yields the solution $x = (\frac{3}{2}, 0)$ with an optimum solution of $z_{11} = 6$ since this is lesser than the current bound, this node is pruned by bounds. With this we have come to the end of pruning all the nodes with the list empty and an optimal solution of $\frac{7}{2}$ with $x = (2, 1)$ and the final solved problem is shown in the figure

5.3 indicating the pruned branches (optimality, bounds, infeasibility), optimal solution and variables.

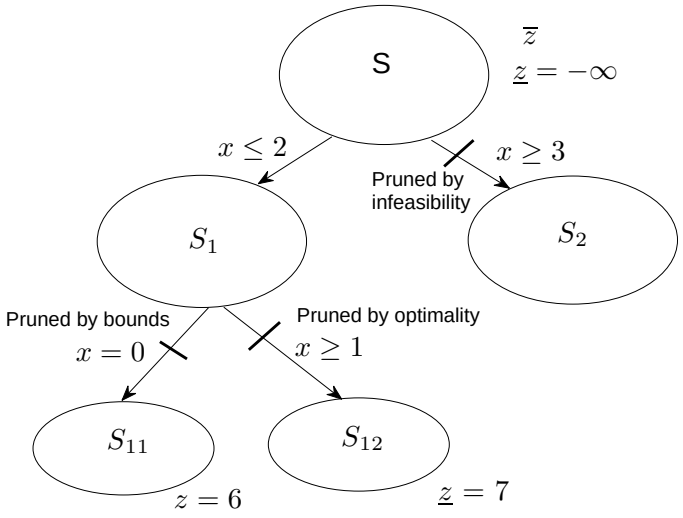


Figure 5.3: Solved problem (pruned branches with optimal solution and variables).

Although this is a very good algorithm to find the optimal solution, with more variables the node size grows larger, handling all the nodes may be inefficient and time consuming. We will now look at the cutting plane algorithm.

5.1.3 Cutting plane scheme

Cutting plane algorithms are vastly used to provide "cuts" in the feasible region to make the constraints more tighter and remove the part of the feasible region which consists of the non-integer points. Since unlike branch and bound (which essentially divides the problem into two sub-problems) cutting plane removes a chunk of the feasible region thus making the problem more tighter to determine the integer solution to the problem. The obvious demerit of this algorithm is that the type of the cut will determine the efficiency of the solution (with respect to computational time and memory), a better cut would imply

that the solution can be achieved faster and vice-versa, we will now deal with one type of a cutting plane algorithm called the Gomory cutting plane algorithm. It should be noted that the cutting plane schemes can be employed after the problem structure is examined and the best cutting plane scheme can be used, here we explain one such scheme although there are numerous other schemes available and depending on the problem type, they can be more efficient and faster, however for illustrative purposes we will consider the gomory cutting plane scheme here.

In the Gomory cutting plane algorithm, the final tableau from the simplex method is used, the final tableau is also referred to the optimal dictionary as it holds the information to construct the solution and its variants. The basic idean in the cutting plane algorithm is to use the separate the fractional and the integer parts and to create a constraint with only the fractional variables involved. Let P and Q be two integers and p and q be two fractional numbers such that

$$P + p = Q + q$$
$$P \leq Q \text{ then}$$
$$p \geq q$$

this is because when we consider the optimal dictionary and have $x_{B(i)} = s$ where s is basic and non integer and there exists a row which is going to have a fractional form [16] as

$$x_{B_i} + \sum_{j \in NB} a_{ij} x_j = b_i$$

where NB is the set of non-basic variables in the simplex tableau. Rewriting the above equation as a function of integer and fractional parts yields

$$x_{B_i} + \sum_{j \in NB} \lfloor a_{ij} \rfloor x_j + \sum_{j \in NB} f(a_{ij}) x_j = \lfloor b_i \rfloor + f(b_i)$$

where $f(a) = a - \lfloor a \rfloor$. Rearranging the terms yields

$$x_{B_i} + \sum_{j \in NB} \lfloor a_{ij} \rfloor x_j = \lfloor b_i \rfloor + f(b_i) - \sum_{j \in NB} f(a_{ij}) x_j$$

We can now write the above equation as an inequality by removing the fractional part on the right hand side,

$$x_{B_i} + \sum_{j \in NB} \lfloor a_{ij} \rfloor x_j \leq \lfloor b_i \rfloor + f(b_i)$$

in the above equation only $f(b_i)$ is the only fractional part, thus the following inequality holds good

$$x_{B_i} + \sum_{j \in NB} \lfloor a_{ij} \rfloor x_j \leq \lfloor b_i \rfloor$$

also since

$$x_{B_i} = b_i - \sum_{j \in NB} a_{ij} x_j$$

we can combine the above the above two equations to obtain an inequality on the fractional part

$$\sum_{j \in NB} f(a_{ij}) x_j \geq f(b_i)$$

this inequality which consists of only the fractional parts is used to construct a constraint which will remove a part of the feasible region and make the problem more tighter and the solution is re-optimized using the simplex method till an integer solution is achieved. The basic steps involved in obtaining a cut are

- **Step 1:** Using the simplex method form the simplex tableau.
- **Step 2:** From the optimal dictionary chose the rows which has a fractional right hand side value.
- **Step 3:** Now separate the variables into integer and fractional part and formulate an equation consisting of only fractional parts (cancel the integer parts) and change the equality ('=') to a greater than sign (' $\geq'$).
- **Step 4:** Now add this cut (constraint) to the above problem and solve it again in the simplex method.
- **Step 5:** If the solution is optimal with only integer values then stop else go to **Step 2**.

  We will illustrate the cutting plane scheme with an example below.

5.1.4 Example for Gomory cutting plane scheme

Let us consider the same example used for the branch and bound scheme [2](shown in eqn 5.2),

$$\begin{aligned} \max z \quad &= \quad 4x_1 - x_2 \\ &s.t. \quad 7x_1 - 2x_2 \quad \leq \quad 14 \\ &\qquad\; 0x_1 + x_2 \quad \leq \quad 3 \\ &\qquad\; 2x_1 - 2x_2 \quad \leq \quad 3 \\ &\qquad\qquad\qquad\quad x_i \in Z, i = 1, 2 \end{aligned}$$

After solving the example with the simplex method yields the result $x = (\frac{20}{7}, 3, 0, 0, \frac{23}{7})$. The results obtained from the final tableau of the simplex method (also referred to as the optimal dictionary) is

$$\begin{aligned} x_1 \quad &+\tfrac{1}{3}x_3 \quad +\tfrac{2}{7}x_4 \quad &= \quad \tfrac{20}{7} \\ x_2 \quad &+x_4 \quad &= \quad 3 \\ -\tfrac{2}{7}x_3 \quad &+\tfrac{10}{7}x_4 \quad +x_5 \quad &= \quad \tfrac{23}{7} \end{aligned}$$

Now $x_1 + \frac{1}{3}x_3 + \frac{2}{7}x_4 = \frac{20}{7}$ is fractional so we will use the gomory cutting plane scheme to provide cuts, separating the fractional and the integer parts as

$$(1)x_1 + (0)x_3 + \frac{1}{3}x_3 + (0)x_4 + \frac{2}{7}x_4 = 1 + \frac{6}{7}$$

Ignoring the integer parts and just retaining the fractional parts and applying gomory's cutting plane inequality we get

$$\frac{1}{3}x_3 + \frac{2}{7}x_4 \geq \frac{6}{7}$$

we will use this cut as a constraint for the original problem, this constraint removes a part of the feasible region which consists of the non-integer optimal solution and thus making the basis move closer to the optimal integer solution. We will call this cut as 'a', where $a = -\frac{6}{7} + \frac{1}{3}x_3 + \frac{2}{7}x_4$, it should also be noted that here a just represents the slack variable obtained from converting the inequality to an equality. After adding this constraint and re-optimizing the problem we arrive at the solution $x = (2, \frac{1}{2}, 1, \frac{5}{2}, 0)$, now since x_2 is non-integer we again consider the tableau of the simplex method and apply cuts to x_2, from the

dictionary,

$$\begin{array}{rrrrcl} x_1 & & & +a & = & 2 \\ x_2 & & -\frac{1}{5}x_5 & +a & = & \frac{1}{2} \\ x_3 & & -x_5 & -5a & = & 1 \\ x_4 & +\frac{1}{2}x_5 & & +6a & = & \frac{5}{2} \end{array}$$

Adding the cut to x_2 again by separating the integer and fractional parts and using gomory cutting scheme to obtain the "cut" we obtain $\frac{1}{5}x_5 \geq \frac{1}{2}$ or $b = -\frac{1}{2} + \frac{1}{5}x_5$. Using this cut as a constraint and re-optimizing it in the simplex method yields the following solution $x = (2, 1, 2, 2, 1, 0, 0)$ which is perfectly integral and yields an optimal integral solution, with $\overline{x} = (2, 1)$.

Cutting planes scheme use very little computational memory and can provide a solution in little time, however when the number of nodes or variables increase then the algorithm will provide cuts which might be identical, thus they may not be removing a significant portion of the feasible region. Again this scheme is similar to branch and bound where instead of the branches (or nodes), constraints are generated. However it is observed when branch and bound scheme when used with the cutting plane scheme yields very good results which are stable and which takes less time to compute. The next section will now deal with the branch and cut scheme.

5.2 Branch and cut scheme

We adopt the following shorthand notation for the binary QP model (4.1) [1]

$$\begin{array}{lrcl} \min & \frac{1}{2}x^TQx + c^Tx & & \\ (s.t.) & Ax & = & b \\ & Bx & \leq & d \\ & x & \in & \{0,1\}^n \end{array} \tag{5.3}$$

for the binary QP model (4.1) that we presented in the previous chapter. Let $Q \in S^n$, $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$,$B \in \mathbb{R}^{p\times n}$ and $d \in \mathbb{R}^p$. We will present a branch and cut algorithm to solve (5.3) to optimality.

We first develop a suitable Lagrangian relaxation model for (5.3). Consider the following

problem

$$\begin{array}{lll} \max\limits_{u} \quad \min\limits_{x} & \frac{1}{2}x^T(Q - Diag(u))x & + \quad (c + \frac{u}{2})^T x \\ (s.t.) & Ax & = \quad b \\ & Bx & \leq \quad d \\ & x & \geq \quad 0 \\ & x & \leq \quad e \end{array} \tag{5.4}$$

where $e \in \mathbb{R}^n$ is the all ones vector. It is easy to show that the objective value to (5.4) provides a lower bound on the optimal value of (5.3). Moreover, one can show that the problem (5.4) is equivalent to the following semidefinite problem

$$\begin{array}{lll} \min & \begin{pmatrix} 0 & \frac{c}{2}^T \\ \frac{c}{2} & Q \end{pmatrix} \bullet \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} & \\ (s.t.) & X_{ii} = x_i \;\; i = 1, \ldots, n & \\ & Ax = b & \\ & Bx \leq d & \\ & \begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix} \geq 0 & \end{array} \tag{5.5}$$

where $X \in S^n$ and the last constraint indicates that the matrix $\begin{pmatrix} 1 & x^T \\ x & X \end{pmatrix}$ is positive semidefinite. We solve (5.5) using a primal-dual interior point method. Let u^* contain the dual variables corresponding to the first n constraints in (5.5).

Consider the following integer quadratic program

$$Z_{BIP} \;\; := \;\; \min_x \{x^T Q' x + c'^T x : \;\; x \in S\} \tag{5.6}$$

where $Q' = (Q - Diag(u^*))$, $c' = c + \frac{u}{2}^*$, and $S = \{x \in \mathbb{R}^n : Ax = b, Bx \leq d, x \in \{0, 1, \}^n$. It is easy to show that (5.6) and (5.3) have the same set of optimal solutions. On the other hand, (5.6) has a convex quadratic function. As a result, the various relaxations of (5.6) will be convex QPs so that can be solved to optimality using primal-dual interior point methods or active set algorithms.

For instance, our initial relaxation in the branch and cut algorithm is

$$\begin{array}{lrcl} \min & \frac{1}{2}x^T(Q - Diag(u^*))x & + & (c + \frac{u^*}{2})^T x \\ (s.t.) & Ax & = & b \\ & Bx & \leq & d \\ & x & \geq & 0 \\ & x & \leq & e \end{array} \tag{5.7}$$

In each iteration of the branch and cut algorithm, we divide the current problem into two subproblems that are easier to solve than the original problem (divide and conquer paradigm). The ith subproblem has the form

$$Z^i := \min_x \{x^T Q' x + c'^T x : \ x \in S^i\} \tag{5.8}$$

where $S^i \subset S$ is the feasible region of the ith subproblem. The quadratic programming (QP) relaxation to the ith subproblem is obtained by relaxing the binary restriction on its variables. In other words, we replace the constraints $x_i \in \{0, 1\}$ with $0 \leq x_i \leq 1$ for all $i = 1, \ldots, n$. We will also add several cutting planes to the QP relaxation of the ith subproblem in step 5 of the branch and cut algorithm. Let $x_{QP}^{i,l}$ denote an optimal solution and $Z_{QP}^{i,l}$ denote the optimal objective value to the QP relaxation to the ith subproblem in the lth iteration of step 5 of the algorithm.

The branch and cut algorithm maintains an incumbent solution x^* in every iteration, denoting a feasible solution to (4.1) that currently has the smallest objective value. Moreover, the algorithm maintains a variable $Z_{UB} = x^{*T} Q' x^* + c'^T x^*$ that currently represents the best upper bound on the optimal objective value to Z_{BIP} to (4.1). The incumbent solution and the upper bounds are updated whenever the algorithm finds another feasible solution with a smaller objective value. When the algorithm terminates, x^* contains an optimal solution and Z_{UB} contains the optimal objective value Z_{BIP} to (4.1).

We present the complete branch and cut algorithm below [1]:

5.2.1 Solve the convex QP relaxation

We solve the convex QP relaxation in step 3 of Algorithm 1 using primal-dual interior point methods or active set methods for convex QPs. Our implementation exploits the sparsity and structure of the QP relaxation. Moreover, we adopt sophisticated strategies

Branch and cut algorithm for binary quadratic programming

1. **Initialize:** *Put the original problem (4.1) on the list of incumbent problems. Set* $Z_{UB} = \infty$, $x^* = \emptyset$, *and iteration count* $k = 1$. *The list can be implemented either as a queue(first in-first out) or a stack(last in-first out).*

2. **List empty:** *In the* k*th iteration , if the list of incumbent problems is empty, then return* x^* *as a optimal solution to (4.1), STOP. Else set* $l = 1$, *remove a problem from the list, label it as the* k*th subproblem and set its first relaxation to be the QP relaxation for this subproblem, and proceed to step 3.*

3. **Solve** k**th subproblem:** *In the* l*th iteration , solve the QP relaxation of the* k*th subproblem and let* $x_{QP}^{k,l}$ *and* $Z_{QP}^{k,l}$ *be an optimal solution and the optimal objective value, respectively. Details are given in section 5.2.1.*

4. **Prune by infeasibility:** *If QP relaxation is infeasible, then prune the* k*th subproblem by infeasibility. Set* $k = k + 1$ *and return to step 2. Else go to step 5.*

5. **Add cutting planes:** *Find cutting planes that cut off* $x_{QP}^{k,l}$, *but preserve the set of feasible solutions to (4.1). Details are given in section 5.2.3. If no cuts are found, then go to step 6. Else add the cuts to the QP relaxation, set* $l = l + 1$, *and return to step 3.*

6. **Prune by bounds:** *If* $Z_{QP}^{k,l} \geq Z_{UB}$, *then prune the* k*th subproblem by bounds. Set* $k = k + 1$ *and return to step 2. Else go to step 7.*

7. **Prune by optimality:** *If all the variables in* $x_{QP}^{k,l}$ *are binary, then update* $x^* = x_{QP}^{k,l}$ *and* $Z_{UB} = c^T x_{QP}^{k,l}$. *Set* $k = k + 1$ *and return to step 2. Else go to step 8.*

8. **Branch on current subproblem:** *Generate two child subproblems* k^1 *and* k^2 *from the* k*th subproblem. Details are given in section 5.2.2. Put these subproblems on the list of incumbent problems. Set* $k = k + 1$ *and return to step 2.*

to warm-start the QP relaxations in every iteration. The idea is to use the solution to the QP relaxation of a subproblem as a starting solution to the QP relaxation of a child subproblem.

5.2.2 Branch on current subproblem

A good branching rule that exploits the problem structure can significantly reduce the number of iterations in the branch and cut algorithm [1]. We describe one such method that is used in step 8 of the Algorithm 1.
If any of the variables in $y^k = x_{QP}^{k,l}$ are not binary and $Z_{QP}^{k,l} < Z_{UB}$. Let y_p^k be a component of y^k that has the largest fractional part. Create two child subproblems k^1 and k^2 from the kth subproblem, whose feasible regions are

$$\begin{aligned} S^{k^1} &= S^k \cap \{x \in \mathbb{R}^n : x_p = \lfloor y_p^k \rfloor\} \\ &= S^k \cap \{x \in \mathbb{R}^n : x_p = 0\} \end{aligned}$$

and

$$\begin{aligned} S^{k^1} &= S^k \cap \{x \in \mathbb{R}^n : x_p = \lceil y_p^k \rceil\} \\ &= S^k \cap \{x \in \mathbb{R}^n : x_p = 1\} \end{aligned}$$

respectively.

5.2.3 Add cutting planes

If any of the variables in the solution $x_{QP}^{k,l}$ to the QP relaxation of the kth subproblem in the lth iteration of step 5 are not binary, then this solution is not a feasible solution to (4.1). We employ heuristics(problem specific in nature) that return cutting planes to cut off $x_{QP}^{k,l}$ but preserve the set of feasible solutions to (4.1). These cutting planes are added to the QP relaxation of the kth subproblem and they strengthen this relaxation (like the gomory cutting plane scheme explained in section 5.1.3. Although, step 5 (adding cutting planes) increases the cost of an iteration, it reduces the total number of subproblems that are enumerated by the branch and cut algorithm, also as stated before, since branch and cut is a combination of a cutting plane scheme and branch and bound they are known to be more robust, less time consuming and more stable [1].

Chapter 6

Implementation and results

6.1 Implementation

Implementing for solving 4.1 to optimality (or feasibility) involves the following three steps.

- **Environment creation using MATLAB^{TM}:** All the parameters required by 4.1 needs to be created. SCW, ECU, BUS, cost communication matrices and miscellaneous randomization are performed with manual inputs (total nodes, total SCWs, total sensors). The shortest distance is computed between all the nodes using Dijkstra's shortest path algorithm for all the nodes. Finally all these parameters are converted into matrices (**Q,c,A,b,s,l,u**) for the general formulation of the QP as in equation 2.1. Figure 6.1 shows the parameters which are randomized based on inputs like total number of nodes, total number of software components, total number of sensors and number of included and excluded software components.

- **CPLEX^{TM} optimization:** After obtaining the matrices, the next step is to invoke CPLEX to optimize the problem. This is done by a MEX function (for MATLAB compatibility) which uses the CPLEX calling library (compatible with C programs only) for accessing CPLEX packages to optimize a given model. Some preprocessing steps are performed to convert the initial matrices obtained into a CPLEX compatible format (this is done to employ sparsity and efficient memory usage). Advanced features in CPLEX to increase the quality of the solution, feasibility or optimal solution

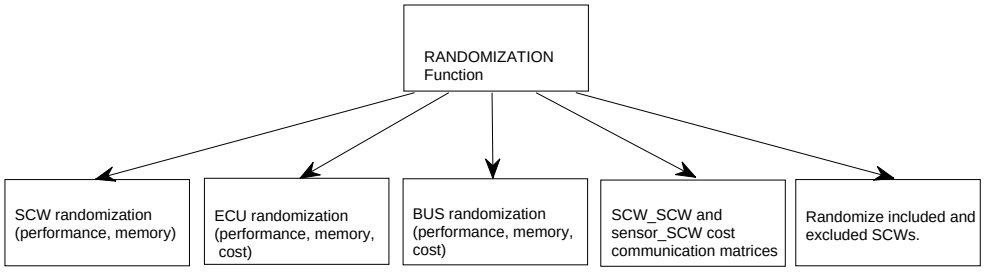


Figure 6.1: Randomization of environment parameters merged as a randomization function.

emphasis and simulation time are set by the user depending on the requirements. Previously suspended simulations can be continued for further optimization by CPLEX till it yields an optimal solution, information regarding total iterations required, cuts generated and tree size memory consumption can be obtained from the log file.

- **Solution files:** The CPLEX results are then displayed at the MATLAB command window. The solution consisting of the optimal solution, optimal values of the variables, slack , solution type, iteration count and simulation duration is stored in a solution file. The solution type determines the type of the solution which can be optimal, optimal with a tolerance gap, feasible and incumbent solution based on time limit, solutions cannot be determined when it is infeasible or unbounded. The problem is written as a QP file (text book format) for ease of readability. From the solution file the optimal variable values are used to plot a graph depicting the nodes which can host ECUs and sensors (and the nodes which cannot) and software component allocation to ECUs using GRAPHVIZ [10].

Figure 6.2 shows a flowchart to implement a software which will solve the given model shown in 4.1. Thus it performs all the randomization, formulates the matrices in the

required CPLEX tool based format, optimizes it, displays the final solution and plots the allocation of software components to the electronic control units.

6.1.1 Testing and validation

Extensive verification of the coefficients, variables and constraints were done using testbenches so that the matrices for CPLEX input were accurate. Some functions like Dijkstra's shortest path were compared using web resources [11] [12] and Graphviz tool for ensuring graph connectivity. For validating the solution for the problems LINDO^{TM}, MATLAB^{TM} optimization package and brute force methods were used for simple examples. Complex examples were tested using a variant of the solutions obtained, (slack values and reduced costs were used to determine the validity of the solution for some examples). The algorithm 5.2 has the potential to return "Certificate of optimality" and thus given enough amount of time, CPLEX will eventually solve the given problem. Considering feasible solutions from a solution pool is a good idea if time is a constraint. Since most of the environment parameters have been randomized, actual values would ensure a better accuracy, however exhaustive testbenches were used to validate the randomized environment parameters to be accurate.

6.2 Results

CPLEX solves a Integer quadratic programming model using branch and bound and also cutting schemes to remove a significant portion of feasible region without compromising on optimality. Figure 6.3 shows the assignment of software component units to ECUs for the example in (3.2). It needs to be noted that if the cost of the ECUs far exceed that cost of the wiring then this model will provide a solution with fewer ECUs and hence more wiring, likewise if the cost of the wiring is far greater than the cost of the ECU, then the solution will contain many ECUs with limited wiring, but the model will always strive to provide a solution which is most optimum i.e. the optimum solution will be the best bound or the best solution wherein the cost of the ECUs and the cost of the wiring is at a minimum for a given problem, thus minimizing the cost of the entire system.

We have solved a range of problems from trivial (such as the example in (3.2))

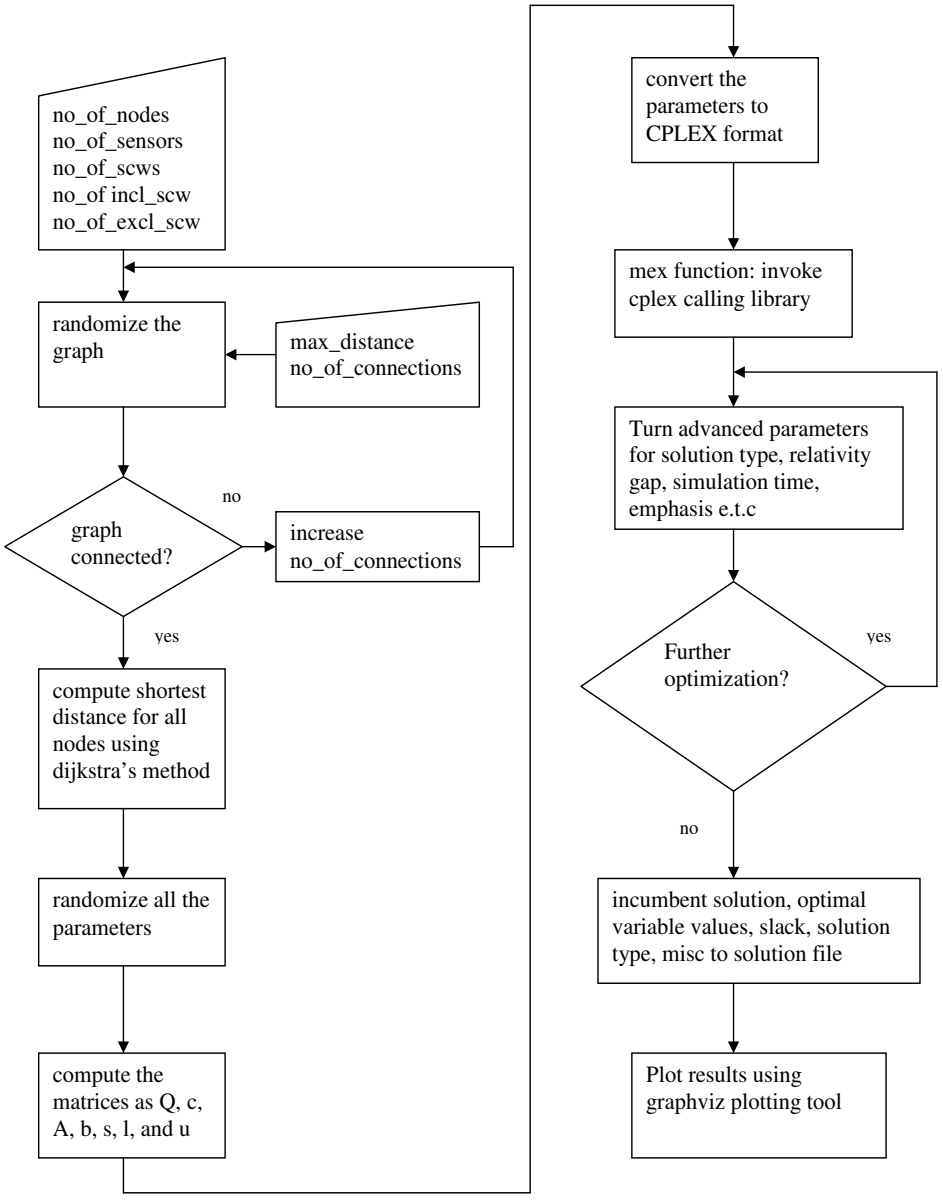


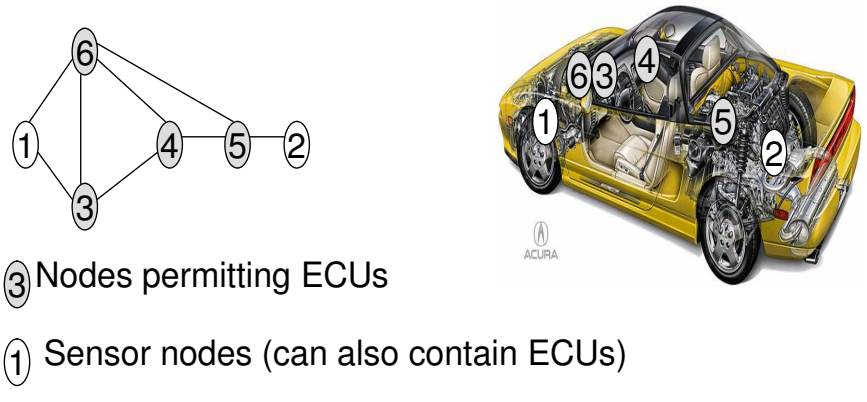
Figure 6.2: Flowchart for the implementation of the software.

to complex, with the increase in the number of nodes, more time and swap space is used for the computations, so emphasis on optimality might not always be the wisest choice, so we shift the solution such that the problem is solved with emphasis on feasibility. In most cases it is observed that even with emphasis on feasibility for large problems , the time and memory used is enormous, thus for practical purposes, the simulation can be stopped with either a time limit or the node size memory limit. The table 6.1 provides a list of benchmarks and the simulation results for a range of problems from a Dual core 2.4 GHz, Intel Core 2, 4GB RAM Linux machine. Some of the problems are of particular importance, benchmark 7 is an approximated model for a real example - HONDA CIVIC LXE 2002 (source of information: Number of sensors and number of ECUs (25) existing in a HONDA CIVIC LXE information provided by John Wilhelmson, automotive mechanic for HONDA motor company, inc), the missing information is approximated such that the example should mimic an actual automobile, information about the sensors and software components is fairly accurate. Benchmark 8 can be considered as a project for a futuristic vehicle, at this juncture we need actual automobile information so that we can perfect the model better and provide more benchmarks for practical applicability.

Results summary for the example

Problem setup:
No of nodes: 6
No of SCWs: 5
No of sensors: 2
Types of ECU: 1
Types of Buses: 2
No of variables: 36
No of constraints: 23
No inclusion and no exclusion properties

SCW:
Performance = (100, 100, 300,50, 200) MIPS
Memory = (200, 100, 500, 100,200) MB



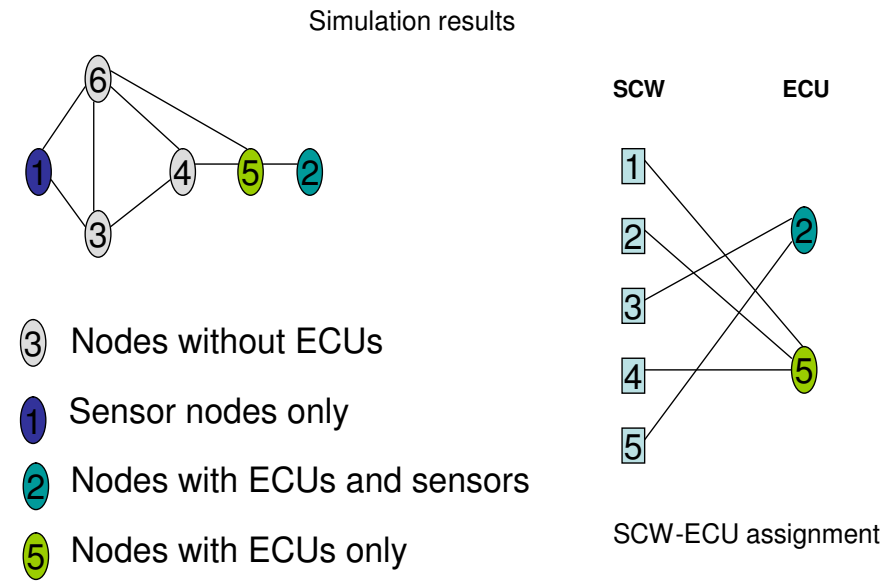


Figure 6.3: Results for the illustrative example.

ECU:

Cost EC1= $20

CPU Performance

EP1 = 500 MIPS

EM1 = 1000 MB

Simulation results

No of ECUs: 2 (at nodes 2 and 5 respectively)

SCW-ECU assignment:

x_{15}: SCW1 assigned to ECU at node 5

x_{25}: SCW2 assigned to ECU at node 5

x_{32}: SCW3 assigned to ECU at node 2

x_{45}: SCW4 assigned to ECU at node 5

x_{52}: SCW5 assigned to ECU at node 2

Table 6.1: Features and simulation results for a list of problems

Problem#	1	2	3	4	5	6	7[1]	8[2]
Number of nodes	6	6	15	40	50	75	100	200
Number of SCWs	5	5	20	50	200	300	500	1000
Number of sensors	2	2	10	15	25	40	50	100
BUS types	2	2	3	6	5	14	10	20
ECU types	1	1	2	5	6	9	15	30
Inclusion elements	-	2	4	10	5	7	10	20
Exclusion elements	-	3	4	5	11	4	15	30
Total Rows (constraints)	23	35	170	2010	900	2175	5400	39800
Total Columns (variables)	36	36	330	2200	10200	23175	51500	206000
Number of ECUs	2	3	5	6	8	17	17	23
Solution type	optimal[3]	optimal[3]	optimal[4]	feasible[5]	feasible[5]	feasible[5]	feasible[5]	feasible[5]
Tree size(CPLEX^{TM})	1 MB	2 MB	22 MB	37 MB	115 MB	160 MB	275 MB	410 MB
Simulation duration	75 sec	120 sec	8 hours	19 hours	27 hours	44 hours	71hours	91 hours

1 2 3 4 5

[1]Problem#7 is an approximated example of a realistic model (Honda Civic LXE 2002)

[2]Problem#8 is an example for a futuristic model

[3]Solution is optimal with certificate of optimality

[4]Solution is optimal with emphasis on feasibility

[5]Solution is the best feasible solution when simulation was terminated due to user defined time limit;

6.2.1 Further Optimization

In this section we will discuss a modification to the binary quadratic programming model (4.1) which will be inherently faster to solve although at the cost of memory. When we consider the model (4.1) the objective function consists of quadratic coefficients, which is the multiplication of two variables. Both the variables are binary and will yield a binary solution, thus the quadratic coefficients can be converted to linear coefficients and thus the binary quadratic programming model can be converted to a binary linear programming model. Although both quadratic programming models and linear programming models [13] have only feasible region (quadratic programming models will also have smooth surface due to linear coefficients), quadratic programming models may have an objective solution anywhere on its surface, if the quadratic programming model is semi-definite convex like 4.1 then there might exist many local minimas, in linear models it is more easier to find the minima because of "flat" surfaces. Thus it is inherently easier to find the optimal solution of a linear programming model than a quadratic programming model. We will now discuss how to convert quadratic coefficients into linear coefficients when the variables are binary [14].

If there exist two variables which are binary x_1 and x_2 which are multiplied in the objective function then introduce a third variable z_{12} such that $z_{12} = x_1 \times x_2$ wherein,

$$0 \leq z \leq 1$$
$$z \leq x_1$$
$$z \leq x_2$$
$$z \geq x_1 + x_2 - 1$$
where z is relaxed between 0 and 1

Although setting up the constraints will require some simulation time, the end result is a linear programming model which can be faster solved. However when we have quadratic coefficients of the order 10000, then the above method will require significant amount of memory as this needs to be done for all the quadratic terms. Thus there is a trade-off between system memory and optimization time, if there is no limit on system memory then the entire environment can be setup as a binary linear model and the optimization to determine the optimal solution by CPLEX^{TM} will be much faster than the actual formulation as

time limit set by user as no new incumbent solution was found for a 12 hour duration

a quadratic programming model, some post-simulation must also be done to restore back to the variables x_1 and x_2, since the quadratic terms will be $f(z)$ and from the variable z we can obtain variables as x. Figure 6.4 and figure 6.5 shows the comparison of memory and simulation time between the model in chapter 4 and the modified model in 6.2.1 using curve-fitting techniques (best-fit) respectively.

Table 6.2: Features and simulation results using the modified model

Problem#	1	2	3	4	5
Number of nodes	6	6	15	40	50
Number of SCWs	5	5	20	50	200
Number of sensors	2	2	10	15	25
BUS types	2	2	3	6	5
ECU types	1	1	2	5	6
Inclusion elements	-	2	4	10	5
Exclusion elements	-	3	4	5	11
Number of ECUs	2	3	5	6	8
Solution type	optimal[1]	optimal[1]	optimal[2]	optimal[2]	feasible[3]
Tree size(CPLEX^{TM})	3 MB	7 MB	40 MB	130 MB	375 MB
Simulation duration	115 sec	235 sec	4 hours	17 hours	14 hours

1 2 3

[1]Solution is optimal with certificate of optimality

[2]Solution is optimal with emphasis on feasibility

[3]Solution is the best feasible solution when simulation was terminated due to user defined time limit; time limit set by user as no new incumbent solution was found for a 12 hour duration

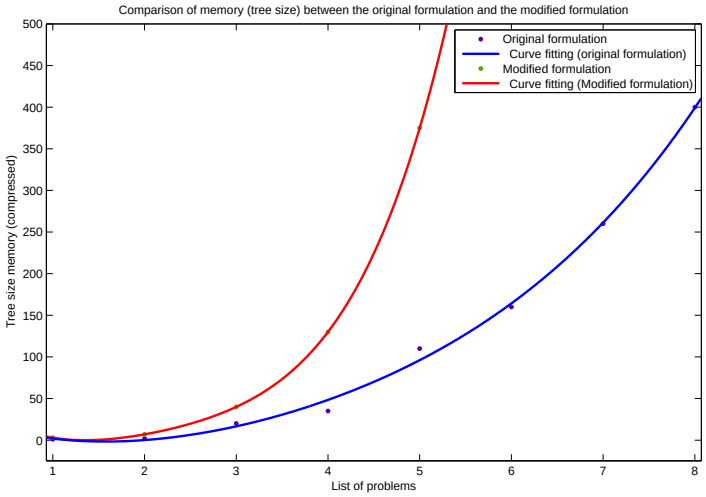


Figure 6.4: Comparison of memory between the formulated and modified models.

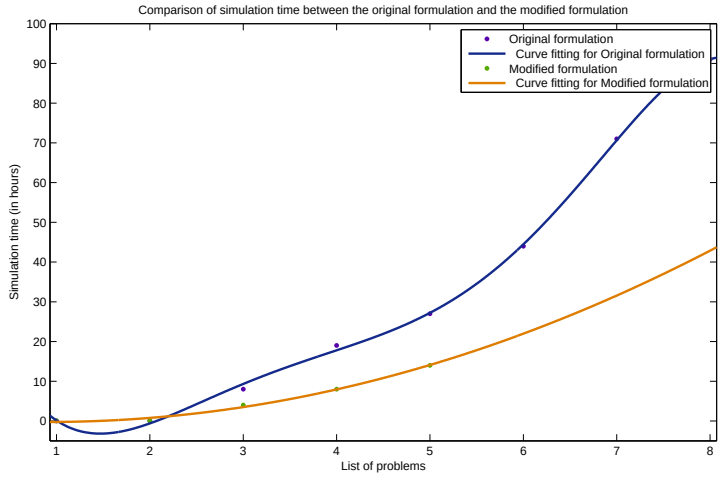


Figure 6.5: Comparison of simulation time between the formulated and modified models.

Chapter 7

Future work and conclusion

7.1 Future work

We have obtained encouraging results from 4.1 and we can further optimize on simulation time by changing the quadratic coefficients to linear ones with a burden on memory using the modified model. With this said, we present some short term aims in the form of problem cleanup and incorporating overlaying nodes and long term goals to consider realistic examples so that our model can be used as a practical application, we will now discuss these goals in the following subsections.

7.1.1 Routing optimization

Solving the model 4.1 to optimality will not yield a direct optimum wiring harness solution, this is because we have assumed that the bandwidth allocated to channels is sufficient and adding additional bandwidth is linear (which is not true), thus the cost communication matrices are only an approximation (logical connections based on shortest distance) and we have to perform a post-optimization step for the physical connections or routing which is applicable to real examples. A two stage process to perform the routing optimization is explained below [1]

1. **Problem setup (assigning ECU to nodes):** This part assumes that bandwidth assigned to a bus, the bus types, software component units accessing the bus type (and hence its bandwidth) is pre-determined based on the shortest path between the

nodes, for example in the illustrative example (section 3.2) it assumes that SCWs 1 and 2 use bus type 1 with cost per meter \$2 and the other pairs which use type 2 with cost per meter \$3, this information is an approximation of the actual cost, since this cost assumes a cost per unit of bandwidth of bus length (for all the bus types). As an illustration, consider the example in section 3.2 , from the results summary for the illustrative example in 6.2 and the results for software component allocation to ECU in figure 6.3, SCWs 3 and 4 are respectively assigned to nodes 2 and 5 (ECUs at the nodes), the shortest route is 1 m as shown on the figure 3.1, and they require 300 MBps of bandwidth between SCWs 3 and 4 (say), and if the average cost of bandwidth is 0.01\$ per 1 Mbps per meter, then the cost which corresponds to SCWs 3 and 4 at nodes 2 and 5 respectively is $PP_{25}^{34} = 300 \times 0.01 \times 1 = \$3$ which is shown in the illustrative example for SCW-SCW communication (section 3.2. This is similar to using an estimate of routing cost at placement stage of place and route.

2. **Problem cleanup (Non-linear cost of adding bandwidth):** In the previous stage we have assumed an approximation of cost , however this approach neglects the nonlinear cost of adding additional bandwidth (i.e adding 1 Mbps of bandwidth to an underutilized channel is free). So far we have assumed that software components have sufficient bandwidth and SCWs assignment to the bus-type (and hence the bandwidth) is determined before the model is solved, we can further optimize the model so that the cost will also consider the bandwidth usage between software components. This can be done as either of the following two steps,

   (a) **Assigning bus types to edges:** We can allocate different bus types to nodes instead of ECU types [1] as

   $$\sum b_{et} BC_t$$

   where $b_{et} = 1$ if bus of type t to an edge e is assigned and BC_t are different types of buses based on their cost and bandwidth, the model can have constraints on bandwidth (similar to memory and performance for ECUs) like

   $$\sum SCW_i to SCW_j - \sum \text{bandwidth at edge e } \leq 0(\text{required bandwidth at an edge e}),$$

   then this approach will minimze the cost due to total bus, gateway and connectors cost by considering the bus types at edges. This approach will be investigated

first.

(b) **Sequential routing** Solving this problem like a sequential routing problem, i.e., sort the SCW links in order of decreasing cost (bandwidth $\times$ shortest path distance). Route them one at a time, so as to minimize the incremental cost of adding that link. This could be done with a Maze router or the A^* algorithm [1].

It is not possible to implement both the stages as a single stage formulation as assigning both ECU to nodes and bus types to edges cannot be done simultaneously and hence assigning bus types to edges can be done as a second stage (problem cleanup) as the software components are already assigned to the respective ECUs. It is important to note that we need to have another stage of problem setup as the bus costs are subjected to change (because problem clean-up may alter the bus types required for SCWs which were pre-determined for problem setup) for better optimization, for example, in the results summary in section 6.2 SCWs 3 and 4 are assigned to nodes 2 and 5 respectively and in the problem clean-up stage a bus is assigned to the edge between nodes 2 and 5 having more than the sufficient bandwidth required, then if another bus type having just enough bandwidth to service SCWs 3 and 4 can be allocated to that edge at a lesser cost then the problem setup has to be optimized again to use this cost to determine if a more optimum solution other than the obtained is present, there might be instances of such back and forth stages (problem setup and problem cleanup) to arrive at the most optimum solution.

7.1.2 Overlaying nodes

In practical problems, there will be a benefit from multiple ECU's or sensors at the same physical location. This is easily handled through a small extension - having multiple nodes in the graph in the same location, e.g two ECU nodes at the same physical location [1]. If the optimum cost benefits from such a solution then this formulation will definitely find it. The cost of such an extension is a growth in the number of nodes and edges, since the cost for two SCWs assigned to a same ECU is zero, overlaying two ECUs at the same node is also a zero.

7.1.3 Features and realistic examples

Although we have presented a system which will automatically setup the model and will optimize the wiring harness, some work is still to be done for realistic automotives. In the course of our definition of the problem statement we might have overlooked into some of the practical limitations on designing such a system for real-life automotives. However we will emphasize that our model is highly flexibly and can incorporate more constraints to covert the model (4.1 from generic to specific, moreover, with the indefinite advantages of model driven designs our software can be "plug and play" compatible. Although we have considered such an example in (6.2), many of the unknown parameters (such as ECU and BUS cost) have been approximated, thus there is a need to optimize an actual system and provide benchmarks for such examples. Our software can be used as a conjunction with virtual prototyping software (eg Mentor's SYSTEMVISION^{TM} or Mathwork's SIMULINK^{TM}) and can be used to analyze the performance and cost of the system and arrive at a solution which can be implemented practically and also be an optimal solution.

We would like to emphasize that the branch and cut algorithm (Algorithm 5.2) is an exact implicit enumeration scheme, i.e., it intelligently sorts through all the feasible solutions of the problem to find an optimal solution. So, when the algorithm terminates, we indeed have an optimal solution to the problem. In many cases, the incumbent solution obtained is already the optimal solution. However, the certification of optimality takes time. Moreover, since the binary linear model is NP-hard, there exist problem instances where Algorithm 5.2 takes several iterations before it terminates.We will also design local search techniques and heuristics to approximately solve our problem. Such algorithms can be applied directly to solve the binary quadratic model (4.1). We will use these solutions returned by these algorithms to validate the quality of the incumbent solution in the branch and cut algorithm, especially when this algorithm does not terminate in reasonable time.

7.2 Conclusion

In this thesis we have offered a precise problem statement for the software component allocation to electronic control units in section (3.1). This statement helps us to set the variables, constraints and the objective function for our model in section (4.1). We present a branch and cut algorithm in section (5.2) that is designed to solve the model to

optimality (branch and cut method is employed by CPLEX). The model is illustrated by a simple illustrative example and successful results obtained from the CPLEX tool, along with certain benchmarks and also provide an alternative modification to the model, when solution time is a bottleneck and system memory is abundant. Finally we conclude with the future work section where we propose methods and strategies to optimize the cost the system by routing optimization and overlaying nodes.

Bibliography

[1] Kartik Sivaramakrishnan and Paul Franzon, North Carolina state university White paper, "Software component allocation to electronic control units", December 2007

[2] Shon Albert, "Solving Mixed Integer Linear Programs Using Branch and Cut Algorithm ", *Masters of Mathematics, NCSU, Fall 2006*.

[3] Cars sag under weighty wiring- 24^{th} October 2005 edition, `http://www.eetimes.com/news/latest/showArticle.jhtml?articleID=172302878`.

[4] Furukawa Yoshimi and Kawamura Seiji, "Automotive electronics system, software, and local area network", *IEEE proceedings of 4^{th} international conference. Hardware/software codesign and system synthesis, 2006*, Seoul, Korea, October 25-26 2006.

[5] Vast Systems technology White paper, "Automotive Electronics: Model-Based Development with Virtual Prototypes", July 2004, `http://www.vastsystems.com/notes/automotive_electronics040727.pdf`.

[6] Model based development with virtual prototypes for advanced features, `http://www.automotivedesignline.com/howto/showArticle.jhtml?articleID=57701843`.

[7] "Aluminum Alloys for Wire Harnesses in Automotive Engineering ", *Springer Wien. BHM Berg- und Httenmnnische Monatshefte,2007*, volume 152, numbers 2-3, pages 62-67, March 2007.

[8] Wayne Winston and Munirpallam Venkataramanan, "Introduction to Mathematical Programming", *4th ed,Brooks Cole : Thomson Learning, c2003*.

[9] Weisstein, Eric W, "NP-Hard Problem." From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/NP-HardProblem.html`.

[10] Graphviz: How to draw dot and neato figures using Graphviz (on-line references), http://www.graphviz.org/Documentation.php.

[11] Java applet by Carla Lafra, PACE university, http://www.dgp.toronto.edu/people/JamesStewart/270/9798s/Laffra/DijkstraApplet.html.

[12] Dijkstra's algorithm Java applet demos, http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/dijkstra/DijkstraApp.shtml?demo1.

[13] Solving linear and quadratic programming models, http://www.solver.com/probtype2.htm.

[14] Multiplication of two binary variables: Linearizing Quadratic coefficients, http://yetanothermathprogrammingconsultant.blogspot.com/2008/05/multiplication-of-binary-variables.html.

[15] Zheng Rongliang and Chen Xiaodong, "Electronic automotive wiring harness", *IEEE proceedings, Vehicle Electronics international conference, 1999* volume 1, pages 443-446, 1999

[16] Kristin Farwell, "GOMORY CUTTING PLANE ALGORITHM USING EXACT ARITHMETIC", *Doctor of philosophy, RPI, Spring 2006*.