A COMPUTERIZED OPTIMIZATION METHOD FOR TOLERANCE CONTROL

by

Junping Yue

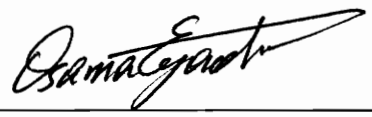
Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

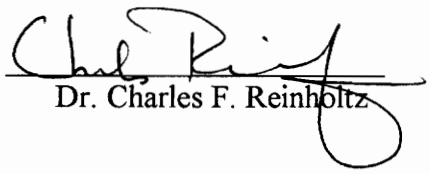
in

Industrial and Systems Engineering

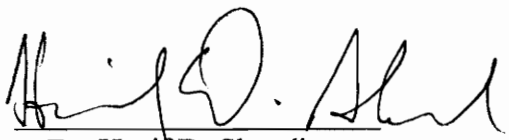
APPROVED:



Dr. Osama K. Eyada, Chairman



Dr. Charles F. Reinholtz



Dr. Hanif D. Sherali

November 1993
Blacksburg, Virginia

A Computerized Optimization Method for Tolerance Control

by

Junping Yue

Committee Chairman: Dr. Osama K. Eyada
Industrial and Systems Engineering

(ABSTRACT)

A systematic optimization method for generating the tolerance chart of a machining part and a computer program to implement this method have been developed. The developed method consists of four major components: the representation scheme for the machining sequence of the part, the algorithm for the identification of tolerance chains, the optimization model for tolerance allocation, and the calculation technique for working dimensions. The tree representation is utilized for capturing the relationships of the machining sequence. Based on this representation scheme, an automated algorithm for identifying the tolerance chains of blueprint dimensions and stock removals is developed. The optimization model consists of two stages: primary and secondary tolerance allocation. The primary stage deals with the optimization of the tolerances of the working dimensions affecting the blueprint dimensions, while the secondary stage deals with the optimization of those tolerances affecting the stock removals. For calculating the working dimensions, a new method is developed to determine the signs of the working dimensions in the tolerance chains automatically. The software is developed using the C programming language. The computerized optimization method has been tested on two examples. In both cases, better results were found when compared to those obtained by a manual method or other optimization approaches.

To my father

ACKNOWLEDGMENTS

I would like to express my sincere thanks and heartfelt gratitude to Dr. Osama K. Eyada for his guidance, patience and support during this work. Sincere gratitude is also extended to Dr. Hanif D. Sherali and Dr. Charles F. Reinholtz for their guidance and suggestions and for serving as members of my graduate committee.

I would also like to thank Ms. Joni Chamber for proofreading my thesis and for her administrative assistance during the period of my study.

I am very grateful to Mr. John Zia for his help to realize my dream.

I would like to thank my father-in-law and mother-in-law for their support and understanding for the past several years.

I am greatly indebted to my parents and my sisters for their love, support and sacrifice that have enabled me to pursue my goals.

Finally, I have my deepest thanks to my wife, Yu Liu, for her love, sacrifice and optimism that gave me great power during this effort. Without her help and consistent encouragement, I would not have completed this degree on time.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Tolerance Control Background

Due to the inherent variations in manufacturing processes, tolerances have been specified on design drawings starting from the late 1920s [Tipnis, 1988]. A tolerance can be viewed as the allowable error or deviation from a theoretically perfect shape or size. Since then, tolerances were used on engineering drawings without any common base until the ANSI Y14.5 tolerance standard was developed to standardize the symbols, the format and their interpretation.

The quality of a product is highly dependent on tolerances specified on its components. The higher the quality of a product, the smaller the tolerance values of its components [Chang, 91]. However, from a manufacturing perspective, tighter tolerances require more careful production procedures and rigorous inspections. Hence the tighter the tolerance, the higher the manufacturing costs. Therefore, in tolerance specification both product functional quality requirements and manufacturing costs must be considered.

Based on the design specifications of a part, the manufacturing engineer has the task of developing a process plan that will detail the manufacturing processes to be used and their sequence. The selection of processes is based on matching design requirements, such as the size and shape of the part, the required tolerances and surface finish, the

material from which the part is to be made, and the required quantity, against the processing capabilities of the available manufacturing facility. These activities are technically called process planning.

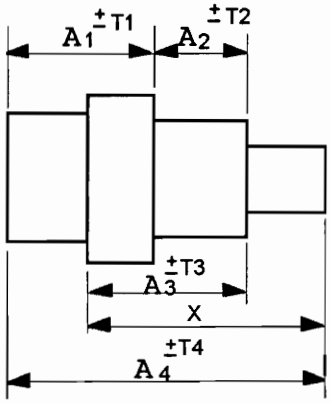
Normally, some of the design specifications can be achieved in a single setup while others would require visiting several setups to be accomplished. For those specifications that require several setups to be achieved, tolerance stacking or accumulation can become a problem. For example, a resultant specification or dimension will be the resultant of a set of working dimensions. This set of working dimensions can be divided into two groups: the positive group whose increase will increase the resultant dimension; and the negative group whose increase will decrease the resultant dimension. If the resultant dimension is denoted by x, it can be calculated as follows:

$$x = \sum_{i=1}^{n} s_i A_i$$

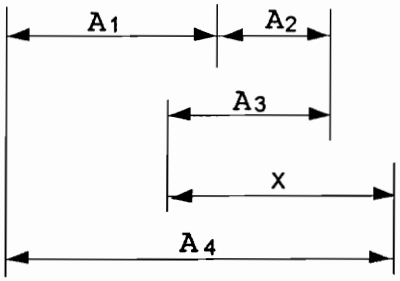
Where:

n	number of working dimensions in the chain,
s_i	equals +1 if A_i is a positive component of x, and
	-1 if A_i is a negative component of x, and
A_i	i-th working dimension in the chain.

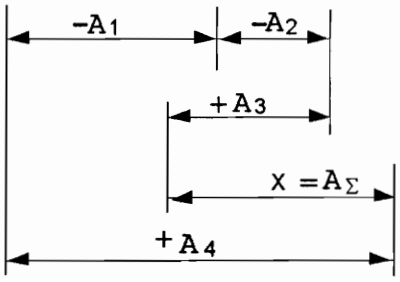
For the example shown in Figure 1.1a, dimension x requires four operations to be achieved. The dimensional chain is shown in Figure 1.1b, and it is composed of five components: A_1, A_2, A_3, A_4 and x. In essence, a dimension does not have the positive or negative nature. Nevertheless, if we take x as the resultant dimension (A_Σ), the rest of



(a) Part drawing



(b) Dimensional chain



(c) Dimensions with signs

Figure 1.1 Dimension Chain and Tolerance Accumulation

components will possess their positive or negative nature as shown in Figure 1.1c. For this case, A_3 and A_4 have positive signs, and A_1 and A_2 have negative signs. Otherwise stated, A_3 and A_4 belong to increasing working dimensions, and A_1 and A_2 belong to decreasing working dimensions.

The produced resultant dimension will fluctuate within a certain range dependent on the particular equipment utilized in producing the dimensions in the chain. An acceptable variation value for the resultant dimension will be its tolerance, which can be denoted by T. To calculate the tolerance of the resultant dimension, all the tolerances of the dimensions in the chain are added, regardless of whether the working dimension belongs to the increasing or decreasing group. This can be written as follows:

$$T_{\Sigma} = \Sigma T_i$$

Where:

T_{Σ} tolerance of the resultant dimension;

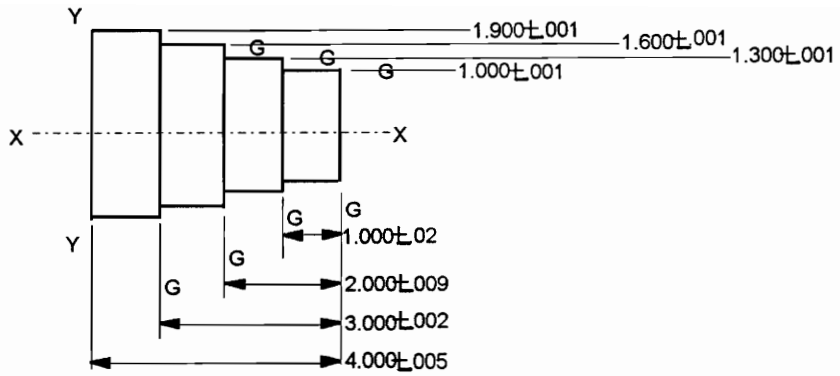
T_i tolerance of the ith component dimension.

This phenomenon is referred to as tolerance accumulation or stacking. The effect of tolerance stacking must be analyzed or controlled before production to avoid costly scraps. Additionally, manufacturing costs can be reduced by better distribution of allowable variations among the utilized processes based on their processing capabilities. These activities are known as tolerance control. Several methods have been developed for tolerance control. The most elaborate and complete method is the tolerance chart, which is described in the next section.

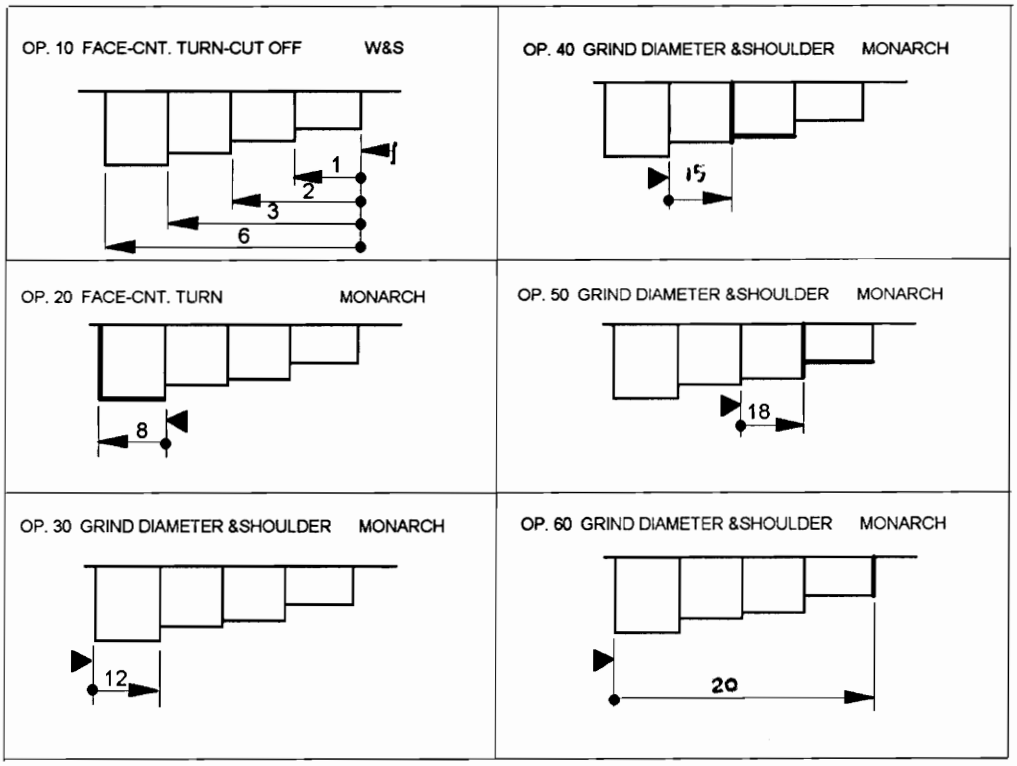
1.2 Tolerance Charts

A tolerance chart is a graphical representation of the sequence of the machining operations that the part has to undergo to be manufactured according to the process plan. The main inputs for tolerance chart development are a part drawing and a tentative process plan. Figure 1.2a shows the blueprint of a steel plug, and Figure 1.2b details its tentative process plan. Figure 1.3 presents the tolerance chart for this part, and Figure 1.4 provides the significance of some symbols used in the chart.

As shown in Figure 1.3, the basic structure of the tolerance chart consists of a part drawing and a chart. The part drawing is located at the top center of the chart, and dimension lines are extended into the chart. The chart has 10 columns, not including the extended section of dimensional lines. The first column is the ordinal numbers of the chart row. The second column provides the operation number. A total of six operations are needed to manufacture this part. The third column describes the name or type of the machine to be used. The fourth and fifth columns are for recording the working dimensions and their tolerances respectively. The working dimension defines the mean value of the dimension at that operation; while the working tolerance gives the maximum allowable variation from the mean dimension. The magnitude of the working tolerance is first set equal to the processing capabilities of the equipment to be used. These values are later relaxed, if one or more of the tolerances of the resultant dimensions are much tighter than those of the blueprint. The relationships between the part surfaces are shown on the extended section of the dimensional lines. The heavy black lines with an arrow at one end and a dot at the other end represent machining cuts; while the heavy black lines with a dot at each end represent balance dimensions. A dot indicates a locating or datum surface and

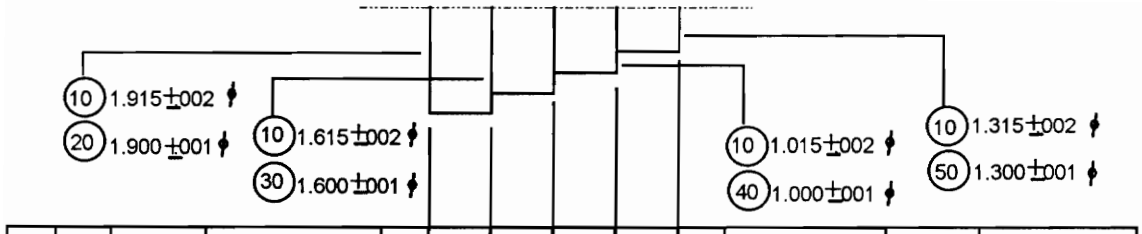


(a) Blueprint of a Steel Plug



(b) Tentative Process Plan

Figure 1.2 Blueprint and Tentative Process Plan



Line No	Oper No	Machine used	Machine to Mean	Machine to +/-tol	Balance dim Mean	Balance dim +/-tol	Line involved	Stock Removal Mean	Stock Removal +/-tol
1	10	W&S	.979	.003				Solid	
2	10		1.994	.003				Solid	
3	10		3.003	.003				Solid	
4					1.009	.006	3-2		
5					2.024	.006	3-1		
6	10		4.031	.010				Solid	
7					1.028	.013	6-3		
8	20	Monarch	1.008	.004			7-8	.020	.017
9					2.017	.010	4+8		
10					3.032	.010	5+8		
11					4.011	.007	3+8		
12	30	Norton	1.000	.001			8-12	.008	.005
13					1.017	.011	9-12		
14					2.032	.011	10-12		
15	40	Norton	1.000	.003			13-15	.017	.014
16					2.000	.004	12+15		
17					1.032	.014	14-15		
18	50	Norton	1.000	.015			17-18	.032	.029
19					3.000	.019	16+18		
20	60	Blnchrd	4.000	.001			11-20	.011	.008
21					3.000	.002	20-12		
22					2.000	.005	20-16		
23					1.000	.020	20-19		
24									
25									
26									
27									
28									
29									
30			BLUEPRINT		RESULTANTS				
31			4.000	.005	4.000	.001	20		
32			1.000	.020	1.000	.020	23		
33			2.000	.009	2.000	.005	22		
34			3.000	.002	3.000	.002	21		
35									
36									

Figure 1.3 An Example of a Tolerance Chart (Adopted from Wade)

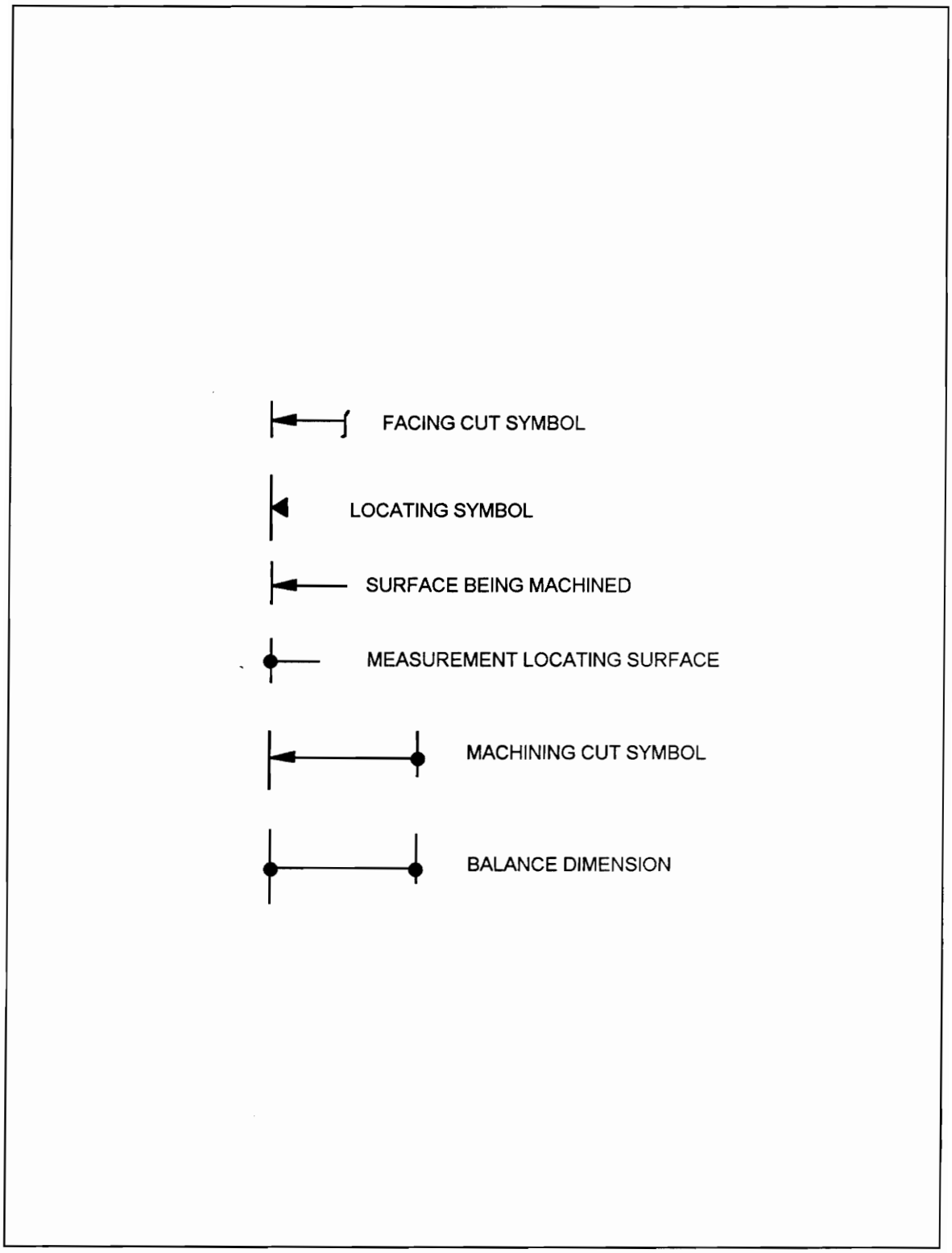


Figure 1.4 Symbols Used in the Chart

an arrow represents a newly generated surface as shown in Figure 1.4. The sixth and seventh columns record the intermediate dimensions, called balance dimensions. A balance dimension results from subtracting or adding two machining cuts or a balance dimension and a machining cut. The eighth column records the line numbers of the two elements composing each balance dimension. The only objective of columns six, seven and eight is to provide a space for manual calculations. The ninth and tenth columns store the amount of stock to be removed at a given operation and its tolerance. It should be noted that the chart contains only the operations in the process plan that have a direct effect on tolerance stacking.

Ten operations will be performed on the part on six setups to achieve the design requirements. These operations correspond to line numbers 1, 2, 3, 6, 8, 12, 15, 18, 20. The line numbers 4, 5, 7, 9, 10, 11, 13, 14, 16, 17, and 19 are for the manual calculation of the balance dimensions. At the bottom of the chart, the required part dimensions and their tolerances are shown on the lower left corner of the tolerance chart under the title BLUEPRINT. The final machining results are given under the title RESULTANTS on the lower right corner of the chart. For the process plan to be feasible, the tolerances of the resultant dimensions must be either equal to or less than the blueprint tolerances; otherwise, the process plan must be revised. However, if the tolerances of the resultant dimensions are less than those of the blueprint, the tolerances of the working dimensions need to be relaxed to reduce manufacturing costs. This process is known as tolerance allocation.

To determine the optimal tolerances for the working dimensions, the tolerance chain of each blueprint dimension (resultant dimension) needs to be identified. The criteria

is that the summation of the tolerances in the tolerance chain of each resultant dimension must not exceed the design tolerance of that dimension, and the operational (working dimension) tolerance must not exceed the designated process capability. For example, the tolerance chain of the blueprint dimension DE is composed of operations 12, 15, 18, and 20. Its tolerance chain can be manually identified from the tolerance chart and it is shown in Figure 1.5. The design tolerance of DE is ±0.020. If the working dimensions of this chain are not in any other resultant dimension chain, the problem of allocating tolerances to the working dimensions is simple. However, the problem of allocating tolerances to working dimensions of parts with several interrelated tolerance chains is complicated, since several tolerance chains need to be examined simultaneously to determine the operational tolerances effectively. Therefore, with the exception of tolerance allocation, the steps of developing the tolerance chart manually are simple. For more detail on the manual tolerance chart method, the reader is referred to two references [Wade, 1967, 1983].

The main benefit of using the tolerance chart is the reduction of manufacturing costs. First, it verifies the accuracy of the process plan, and in turn prevents costly scrap. Second, it reduces inspection costs through the relaxation of the tolerances of the working dimension. Finally, it can be used to determine the proper raw material size.

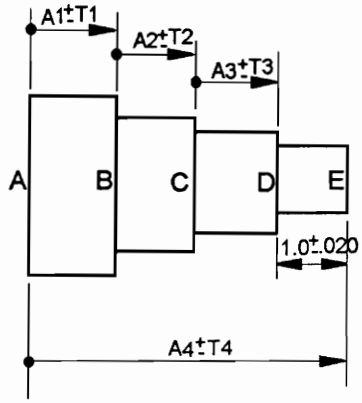


Figure 1.5 The Tolerance Chain of the Dimension DE

1.3 Automation of Tolerance Chart

Although the tolerance chart was developed in the 1950s, its applications have been very limited. Few applications can be found in the aircraft and the automobile industries [Ahluwalia, 1984; Whybrew, 1990]. This is due to the complexity and the manual nature of the tolerance chart method. Creating a tolerance chart is tedious, time-consuming, and prone to mistakes. Also, the optimum tolerance values for the working dimensions can not be determined manually. With the widespread use of computers in the 1980s, various automated methods were developed to automate some of the tolerance charting aspects.

Some research efforts have concentrated on a representation method of the machining sequence along with a method for determining the tolerance chains. Xiaoqing and Davies [1988] described a matrix-tree-chain representation method, where rows of the matrix correspond to the line numbers and its columns correspond to the extended dimensional lines of the manual tolerance chart method. Several processing steps are performed to first convert the matrix to a square one and to then determine the trees from which the tolerance chains can be identified. Irani et al. [1989] introduced a directed graph representation method along with a set of complex algorithms to determine the tolerance chains. Complex algorithms were needed because of the inadequate representation of all the required information for easier processing. For example, although more than one machining cut will take place on a given surface, only one node is used to represent it. Whybrew et al. [1990] developed a tree-based representation method along with a manual tabulated method for the identification of tolerance chains. Again the representation did not include all the information required for automatic identification of

tolerance chains. Ji [1993] also utilized a tree and described algorithms to develop the tree in three steps along with a manual method for the identification of tolerance chains. Although the final tree provides a complete representation of the machining sequence, the tree can be developed straight from the process plan and without any need for these algorithms.

Other research efforts focused on developing optimization methods for the allocation of tolerances among the working dimensions. Xiaoqing and Davies [1988] presented an iterative search method. However, the method does not include the processing capabilities of the equipment to be used, and, in turn, a realistic framework for optimization does not exist. Irani et al. [1989] introduced a linear programming model for tolerance allocation. However, the model does not distinguish between the tolerance chains of resultant dimensions and stock removals for tolerance allocation. The working dimensions affecting only the tolerances of stock removals can be relaxed more than those directly affecting the chains of the resultant or design dimensions. Also, the model does not consider all the required constraints, such as the maximum and minimum allowable stock removals for each machine.

Finally, some research efforts were directed towards developing software for computer-aided tolerance charting. A computer aided tolerance control system (CATC) was developed by Ahluwalia and Karolin [1984]. The system consists of three modules: input, calculation processing, and output. The input module provides graphical capabilities for the user to input required drawings, such as part and raw material drawings, and the process plan information including dimensions, tolerances, stock removals and processing capabilities. A database of stock removals and processing

capabilities is also provided. The calculation processing module determines working dimensions, stock removal tolerances, and resultant dimensions with their tolerances. The output module prints the manual tolerance chart automatically. However, an optimization method for tolerance allocation is not included in the system. Also, it does not have the intelligence to handle any violation of constraints. When a violation occurs, the user is prompted to loosen the tolerances on the working dimensions.

Xiaoqing and Davies [1988] computerized their matrix-tree-chain representation method and their iterative search approach for tolerance allocation. The software is written in FORTRAN and runs on a VAX 11/750 under the VMS operating system. As indicated earlier there are some drawbacks to their work.

1.4 Problem Statement

As indicated earlier, the manual tolerance chart method is the most elaborate and complete tool for tolerance control. However, creating a tolerance chart manually is tedious, time-consuming, and prone to mistakes. Also, the method is not efficient in determining the optimum tolerance values for the working dimensions when the number of tolerance chains that need to be analyzed simultaneously is large. Finally, an automated tolerance charting method is needed for computer-aided process planning so that tentative process plans can be verified and optimized.

Research efforts to automate tolerance charting have concentrated on one or more of the following: (1) a representation scheme along with an algorithm for identifying

tolerance chains, (2) an optimization method for tolerance allocation, and (3) software development. The main drawbacks to current representation and tolerance identification schemes include: (a) the manual tolerance charting method must be performed first, (b) the tolerance chains are manually determined, and/or (c) complex algorithms are required to determine the tolerance chains due to inadequate representation. The main drawbacks to present tolerance allocation techniques include: (a) incomplete representation of processing capabilities, and/or (b) lack of treating the tolerance chains of resultant dimensions and stock removals separately. The working dimensions affecting only the tolerances of stock removals can be relaxed more than those directly affecting the chains of the resultant dimensions, and thus lower manufacturing costs can be achieved. Finally, existing tolerance charting software does not include an efficient optimization method for tolerance allocation.

From the above, it can be concluded that a complete, efficient and computerized method for tolerance control needs to be developed. Ideally, this method should consist of the following:

1. A complete and computerized representation scheme of the machining sequence.
2. An algorithm for automatically identifying the tolerance chains from the representation scheme.
3. A computerized optimization method for tolerance allocation among the working dimensions.
4. Algorithms to determine all the machining parameters for each operation automatically.

The representation method is the basis for any automated tolerance charting efforts. It must contain the major input information for developing the tolerance chart, and allow for identifying tolerance chains efficiently. Identifying the tolerance chains is the first step to control and analyze the accumulation of tolerances during the manufacturing stages of a part. The optimization method for tolerance allocation should distinguish between the optimization of tolerance chains of resultant dimensions and those of stock removals. Also, the optimization process must be constrained by all the processing capability parameters. Finally, the determination of operational dimensions and their tolerances for each machining operation in the process plan is the purpose of the tolerance chart. All these calculations must be performed automatically.

1.5 Research Objective

The objective of this research is to develop a computerized optimization method for tolerance control. The developed software, written in the C programming language, includes the following features:

1. A tree representation of all the information in the machining sequence, along with a procedure for creating the tree from the tentative process plan.
2. An algorithm for identifying tolerance chains using the predecessor index list.

3. A two stage linear programming optimization method for tolerance allocation. The first stage deals with the tolerance chains of resultant dimensions, while the second stage addresses those of the stock removals.
4. Computerized algorithms to calculate the working dimensions from the tree representation.

1.6 Thesis Outline

Chapter 2 first presents some basic graph concepts utilized in the representation methods for capturing the machining sequence. A detailed literature review is then provided, and followed by a summary of previous work and research needs.

Chapter 3 first presents the developed tree representation scheme for capturing the machining sequence. An algorithm for finding the tolerance chains using the predecessor index list is then described along with an example. The principles of tolerance allocation are then outlined, and followed by the developed optimization model for tolerance allocation along with an example. In the last section, the automated algorithms for calculating the working dimensions based on the tree representation are detailed.

Chapter 4 describes the developed computerized optimization system in terms of the input file, program structure, and output format. The results obtained from the developed method on two examples, commonly found in the literature, are then provided. Finally, Chapter 5 provides some concluding remarks and suggestions for future research works.

CHAPTER 2

LITERATURE REVIEW

Although tolerance charts appeared in the early 1950s as a tool for reducing the manufacturing costs of machining parts, the technique has not been widely used in industries because of the amount of time and effort required in applying it. Owing to the advances in computer technology and computer-aided process planning in the 1980s, several research efforts were directed to automate the process of tolerance charting. These efforts have mainly concentrated on one or more of the following aspects: (1) representation scheme and an algorithm for identifying tolerance chains, (2) tolerance allocation method, and (3) computer implementation.

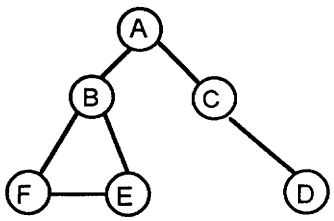
This chapter first presents some relevant graph concepts, since it is the most widely used representation method of the machining sequence. Previous work on the automation of tolerance charting is then reviewed along with the advantages and disadvantages of each approach. The chapter concludes with a summary of previous work and current research needs.

2.1 Basic Concepts of Graphs

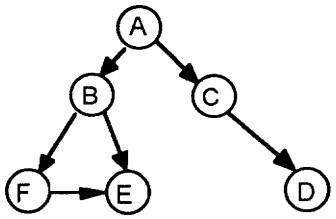
A graph is a set of nodes connected by a set of arcs, where each arc is specified by a pair of nodes (see Figure 2.1a). The set of nodes is A, B, C, D, E and F and the set of

pairs is (A,B), (A,C), (B,E), (B,F), and (C,D). Since no relations exist between the elements of each pair, these pairs are considered unordered. For example, the pair (A,B) can also be (B,A). If the pairs of nodes that make up the arc are ordered pairs, the graph becomes directed, or called digraph (see Figure 2.1b), and the set of ordered pairs is (A,B), (A,C), (B,E), (B,F), (C,D), and (F,E). Each node in a directed graph has an indegree, number of arrow heads entering the node, and an outdegree, number of arrow tails leaving the node. For example, node B in Figure 2.1b has an indegree of one and an outdegree of two; while node D has an indegree of one and a zero outdegree. A path is defined as a sequence of nodes and arcs with the beginning and ending at nodes. For example, the path from node A to node D in Figure 2.1b is {A, (A, C), C, (C, D), D}. When there is no ambiguity, the path is denoted by {A, C, D}. If the start and end nodes of a path are the same, the path is a closed one, or called circuit. A chain is similar to a path except that not all arcs have the same direction. A cycle is a closed chain. If a graph contains a cycle, it is cyclic; otherwise it is acyclic.

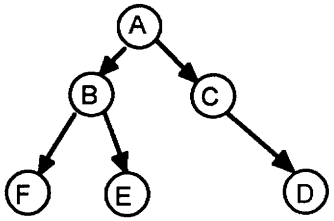
A tree is defined as a connected graph with no cycles, in which only a single node, called the root, has a zero indegree and every other node has an indegree of one. This implies that any node in the tree except the root has exactly one node above it, which is called its predecessor. The nodes directly below that node are called its descendant. In Figure 2.1c, node A is the root of the tree and nodes D, E, and F are its descendants. Any node in a tree that does not have descendants, with an outdegree equal to zero, is called a leaf node and any node that has both the predecessor and descendant(s) is an interior node. For example, in Figure 2.1c, nodes F, E, and D are leaf nodes and nodes B and C are interior nodes. The most important property of a tree is that there is exactly one chain connecting any two nodes.



(a) Undirected Graph



(b) Directed Graph



(c) Tree

Figure 2.1 Graph Types

2.2 Computer Aided Tolerance Charting

One of the important functions of the tolerance chart is the identification of the dimensional chains of resultant dimensions and stock removals to control the stacking of tolerances during manufacturing. The conventional method utilizes a set of manual rules to detect these dimensional chains, which is rather complicated and time consuming. The following sections describe previous research efforts in automating the process of tolerance charting to overcome some of these disadvantages. Section 2.2.1 provides different representation methods along with their approaches for the identification of tolerance chains, Section 2.2.2 describes the tolerance allocation methods, and Section 2.2.3 details the computer implementation efforts.

2.2.1 Representation Methods

Previous representation methods can be classified into three groups: matrix-tree-chain, directed graph, and tree-based methods. These methods are described in detail below.

2.2.1.1 Matrix-Tree-Chain Method

Xiaoqing and Davies [1988] presented a matrix-tree-chain method to automate the manual trace procedure developed by Wade [1967, 1983] for the identification of tolerance chains, and to allow for the automatic calculation and adjustment of tolerances. Each row of the matrix corresponds to either a working dimension or a resultant

dimension. The rows or dimensions are arranged according to their order in the manual tolerance chart method. Also, each column corresponds to a vertical line or surface in the manual tolerance chart method. The elements of the matrix are either 0, 1 or 2. For the working dimension, the machined surface is denoted 2, and the surface measured from is denoted 1. For the resultant dimension, both surfaces are denoted 1. Otherwise, a zero is used. A series of matrix operations are applied to first convert the matrix to a square one. Sub-trees are then identified from the matrix and tree operations are applied to determine the tolerance chains for both resultant dimensions and stock removals.

The main drawback of this representation method is that the manual tolerance chart method must be first performed to determine the working dimensions and their appropriate order on the chart with disregard to their relevance for automation. Also, the method captures all the information provided in the chart. Some of this information is redundant for the automatic identification of tolerance chains [Ji, 1993].

2.2.1.2 Directed Graph Method

Irani et al. [1989] introduced a directed graph method to represent the relationships between the locating and working surfaces. A directed graph representation of the machining sequence of the example described in Chapter 1 is shown in Figure 2.2. The lettered nodes represent surfaces, while the numbered arcs represent the machining cuts between surfaces. The tail node of an arc represents the locating surface for the machining cut, while the head node corresponds to the working surface on which material

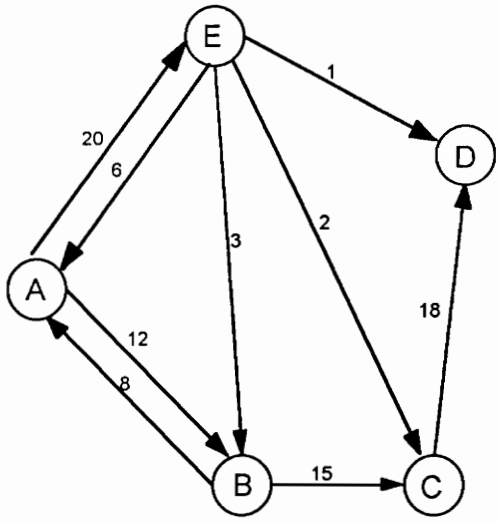


Figure 2.2 A Directed Graph Representation of Machining Sequences

removal occurs. A set of complex algorithms were developed to determine the tolerance chains for both resultant dimensions and stock removals.

The main drawback of this representation method is the inadequate representation of all the required information for easier processing. For example, although more than one machining cut will take place on surface A, only one node is used to represent this surface. For this reason, complex algorithms are needed to take care of this inadequate representation.

2.2.1.3 Tree-Based Methods

Whybrew et al. [1990] presented a tree method for representing the machining sequence and relationships between the locating and machining surfaces. The initial locating surface is the root node of the tree. Each link represents a machining operation with its associated working dimension, and each node represents a machined and/or a locating surface. Figure 2.3 shows a tree representation of the example described in Chapter 1, where the root node is the initial locating surface, the leaf nodes are the machining surfaces, and the interior nodes are both locating and machining surfaces. A manual tabulated method is then applied to determine the tolerance chains of both resultant dimensions and stock removals. The main drawback to this representation method is that not all the information required for automatic tolerance allocation is represented. For example, the working dimensions are not captured in the tree.

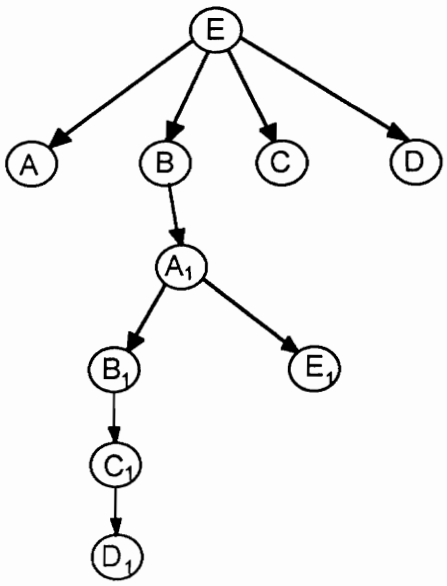


Figure 2.3 A Tree Representation of Machining Sequences

Ji [1993] also proposed a tree representation, where the tree is created in three steps. First, a tree representing the relationships between the blueprint dimensions is created. Based on this tree, a stock removal tree is then developed by adding the stock removals on the appropriate nodes of the blueprint dimension tree. Finally, the working dimension tree is created from the stock removal tree so that tolerance chains can be identified manually. The working dimension tree has the same vertices as the stock removal tree, but it is of the directed tree type. Algorithms for developing each tree and a manual method to determine the tolerance chains are provided. The final tree captured more information than that of Whybrew et al. [1990], because the working dimensions are represented as attributes of the arcs (see Figure 2.4). Actually, this tree included all the information required for the automatic identification of tolerance chains and tolerance allocation. However, the main drawback to this method is the complexity by which the tree is created. The tree can be directly generated from the process plan without any need for all these steps and algorithms.

From the above it can be concluded that the tree-based representation is simple, direct and can capture all the required information to fully automate the process of tolerance charting. The prominent advantage is that it can capture the evolution process of surface features easily. Using this method, a unique dimensional chain for any resultant dimension or stock removal can be detected explicitly. The major advantages of tree over graph representation methods are as follows:

(1) The tree can easily capture the different stages of a surface evaluation.

(2) The tree reflects the precedence of machining processes in a hierarchical order.

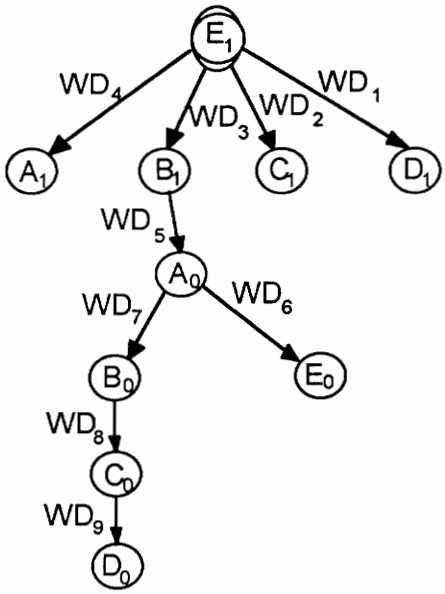


Figure 2.4 Working Dimension Tree

(3) The identification of dimensional chains is direct and does not require complicated algorithms to recognize the path of both resultant dimensions and stock removals.

2.2.2 Tolerance Allocation

Tolerance allocation is the process of determining the appropriate tolerances for the working dimensions based on the design tolerance specifications (i.e., functional requirements) and the processing capabilities of the equipment specified in the process plan (i.e., manufacturing costs). The general principles are as follows:

1. The maximum possible tolerances should be assigned to each machining cut without violating the specified blueprint tolerances.
2. Tolerance values assigned to every cut should be within the processing capability of the machine.
3. Minimum and maximum stock removals for each cut should be feasible with respect to the machine's capability.

Examination of the literature indicates to two previous methods for automatic tolerance allocation: an iterative procedure and a linear programming method. These two methods are explained in detail below.

2.2.2.1 Iterative Approach

A two-step procedure was proposed by Xiaoqing and Davies [1988] for tolerance allocation among the working dimensions:

(1) Tolerance assignment: A tolerance value "t", based on the nominal size of the working dimension (D in mm) and the capability of the process to be used (IT), is assigned to each working dimension. The tolerance value can be calculated using the following formula:

$$t = (.45\sqrt[3]{D} + 0.001D)10^{\frac{IT-16}{5}}$$

(2) Tolerance adjustment: An iterative algorithm is then used to adjust these tolerance values, only when a difference exists between the design and resultant dimension tolerances. The iterative operation is carried as follows:

$$t_i^{(k+1)} = t_i^{(k)} + t_i^{(k)} \times X_{\max}^{(k)} \times S_{ij}^{(k)}$$

Where:

k number of iterations,

X_{max} maximum ratio of difference between the resultant tolerance and blueprint tolerance, and

S proportion of the tolerance of individual dimension to the sum tolerance in the tolerance chain.

The main draw back of this method is the lack of an optimization framework because the processing capabilities are not included in the iterative equation (Irani et al., 1989). Also, the convergence speed of the algorithm is highly dependent on the initially assigned tolerance values.

2.2.2.2 Linear Programming Approach

A linear programming (LP) method was proposed by Irani et al. (1989) to optimize the tolerance allocation among the working dimensions. Its objective function is to minimize the cumulative residuals on all the blueprint dimensions as well as the stock removals. The model is as follows:

Minimize $\sum_{i=1}^{n} z_i + \sum_{j=1}^{m} w_j$

Subject to $\sum_{k \in BP_i} t_k + z_i = b_i \qquad \forall i = 1, 2, ..., n,$

$\sum_{k \in SR_j} t_k + w_j = s_j \qquad \forall j = 1, 2, ..., m,$

$t_k \geq LPC_k \qquad \forall k = 1, 2, ..., p$

$z_i, w_j \geq 0$

where:

z_i residual tolerance in the ith blueprint dimension constraint,

t_k tolerance on the kth machining cut,

BP_i set of machining cuts in the schematic for the ith blueprint dimension,

b_i maximum design tolerance allocated to the ith blueprint dimension,

n number of blueprint dimensions specified on the component drawing,

SR_j set of machining cuts in the schematic for the jth stock removal,

w_j residual tolerance in the jth stock removal constraint,

s_j maximum tolerance on the jth stock removal,

m number of stock removal lines on the tolerance chart,

LPC_k lower limit on the tolerance for cut k defined by the process capability of the machine tool assigned, and

p number of machining cuts in the process plan.

The first set of constraints ensures that the accumulation of tolerances on working dimensions producing each blueprint dimension is less than or equal to the design tolerance specified. The second set of constraints ensures that the accumulation of tolerances on working dimensions related to stock removals is less than or equal to the stock removals. The third set of constraints reflects the influence of the process capability of each equipment.

The limitation of this method is that it does not discriminate between machining cuts that have a direct impact on the tolerance stackup of resultant dimensions and those that only have an effect on the tolerance stackup of the stock removals. Machining cuts that directly influence the resultant dimensions should be constrained more than those affecting the stock removals only. In addition, the maximum permissible stock removals of equipment is not constrained in the model. This may result in stock removal values larger than the capabilities of the equipment.

2.2.3 Computer Implementation

Two computer prototypes, CATC and CADP, were developed to automate some aspects of tolerance charting. These are explained in detail below.

2.2.3.1 CATC

A computer aided tolerance control system (CATC) was developed by Ahluwalia and Karolin [1984] to computerize the manual tolerance chart method. The system consists of three modules: input, calculation processing, and output. The input module provides graphical capabilities for the user to input required drawings, such as part and raw material drawings, and the process plan information including dimensions, tolerances, stock removals and processing capabilities. A database of stock removals and processing capabilities is also provided. The calculation processing modules determines working dimensions, stock removal tolerances, and resultant dimensions with their tolerances. Finally, the output module prints the manual tolerance chart automatically.

The main objective of the CATC system is to only computerize the manual tolerance chart method. An optimization method for tolerance allocation is not included. Also, it does not have the intelligence to handle any violation of constraints. When such conditions occur, the user is prompted to loosen the tolerances on the working dimensions.

2.2.3.2 CADP

Xiaoqing and Davies [1988] presented a computer aided dimensional planning system (CADP). The matrix-tree-chain method, described above, was computerized to detect the tolerance chains of both resultant dimensions and stock removals, as well as the iterative approach for tolerance allocation, also described above. The software is written in FORTRAN and runs on a VAX 11/750 under the VMS operating system. As indicated earlier, the main drawbacks of the matrix-tree-chain method is that the manual tolerance chart method must be first performed to determine the working dimensions and their appropriate order on the chart. Additionally, the iterative approach to tolerance allocation does not have a reliable optimization framework, since the processing capabilities of the equipment are not included in the algorithm.

2.3 Summary and Research Needs

As the literature review indicates, to improve the tedious manual procedure for generating the tolerance chart, research efforts have concentrated on three aspects: representation method, tolerance allocation and computer implementation. The purpose of exploring representation methods for tolerance charting is to attempt to represent the tolerance chain explicitly so as to identify it easily and automatically. The aim of tolerance allocation is to fully utilize the tolerance range allowed by the designer so as to guarantee the most economical manufacturing. The intention of computer implementation is to reduce development time, and to meet the demand for computer aided manufacturing.

Several schemes have been proposed for representing the relationship of machining processes. Among them the most simple and direct method is the tree approach. Not only does it represent the precedence of machining operations explicitly, but it also provides a direct method for identifying the tolerance chain manually. Current tree representation schemes are only suitable for manual identification of tolerance chains. The representation method and algorithm, which not only provide the machining sequence clearly, but also are suitable for computer processing to find the tolerance chain automatically, has not been developed. Further research still needs to be done.

Unlike the representation problem, tolerance assignment or optimization of tolerance allocation still is at its infancy. Although linear programming and iterative methods were presented, the current linear programming model does not take into account some practical constraints in the model, and the iterative method cannot guarantee optimal results. These are the obstacles to realizing the automation of tolerance charting. As for computer implementation, it is a necessary component of a computer aided manufacturing system. To a certain extent, its success depends on the progress of the first two problems.

CHAPTER 3

METHODOLOGY

This chapter presents the methodology for generating the tolerance chart automatically. Section 3.1 describes a tree representation method for capturing the machining sequence along with a procedure to generate the tree based on the tentative process plan. Section 3.2 details how the tolerance chains of blueprint (or resultant) dimensions and stock removals can be determined from the machining sequence tree manually. Section 3.3 first provides some of the definitions of list structures, used to describe the tree on the computer, and then describes an algorithms developed to identify the tolerance chains using the predecessor index list. An example is provided to illustrate the mechanics of the algorithms. Section 3.4 describes the developed optimization model for tolerance allocation along with an example. Section 3.5 first describes how the working dimensions can be determined from the linear equations derived from the tolerance chains, and then details a new method for distinguishing between the signs of working dimensions in a given tolerance chain for automated processing.

3.1 Machining Sequence Tree

A tree is used for representing the relationships between surfaces in the machining sequence. A node would represent a location surface, a working surface or both. A directed arc would represent the relationship between the locating and working surfaces.

The node connected with the tail of the arc is the locating surface, and the node connected with the head of the arc is the working surface. The order number of machining operations is captured as an attribute of the arcs. The first node, the root of the tree, is the initial locating surface of the part. To represent the evolution process of the part surface, the node is labeled with a letter having a subscript. If the node is the root of the tree, the subscript of the node's letter is zero. The machining nodes (surfaces) that are located from the root node are labeled with letters having the subscript 1. After the first operation is completed, the location surface may change. In these cases, one of the nodes with the subscript 1 will become a locating surface. The nodes associated with this new location node are labeled with the subscript 1, if no previous operations were performed on them; otherwise a subscript of 2 is used.

The tree can be directly created from the tentative process plan (assuming each surface has been assigned a letter for identification) using the following developed procedure:

Step 1. Create a root node with a corresponding letter having a subscript 0 to represent the first locating surface.

Step 2. Draw a number of nodes equal to the number of operations to be performed on setup k = 1, where k is an index to the setup number. Each node should be labeled with a letter having the subscript 1. Arcs connecting the root node with the current nodes should be then drawn with arrows pointing towards the current nodes. Each arc is then labeled with the corresponding operation number.

Step 3. Create nodes equal to the number of operations on setup k = 2. The label of each node should have a subscript representing the number of machining times performed on the surface. For example, if the surface was machined once the subscript is 1; if the surface was machined twice the subscript is 2, and so on. Draw arcs between the node (location surface) at the above level and the current nodes with arrows pointing towards the current nodes. Each arc is then labeled with the corresponding operation number.

Step 4. Repeat Step 3 with the setup k = k + 1 until the last setup to complete the tree.

The nodes of the created tree can be classified into initial nodes, intermediate nodes, and final nodes, to correspond to initial, intermediate, and final surfaces of the machining part. Each node is labeled by a letter with a subscript. Nodes with same letters represent same surfaces. The subscripts would reflect the evaluation process of the surface during manufacturing. For example, the subscript number would reflect the number of machining times performed on that surface. Finally, the number of levels in the machining tree should correspond to the number of setups in the tentative process plan.

The machining sequence tree of the steel plug shown in Figure 1.2 was generated using the above procedure and is illustrated in Figure 3.1. E_0 is the initial location surface. A_1, B_1,C_1, and D_1 are initial machining surfaces. A_2, B_2,C_2, D_2, and E_1 are final surfaces. Surface A was processed twice to meet design requirements. A_1 is its initial machining surface, and A_2 is its final machining surface.

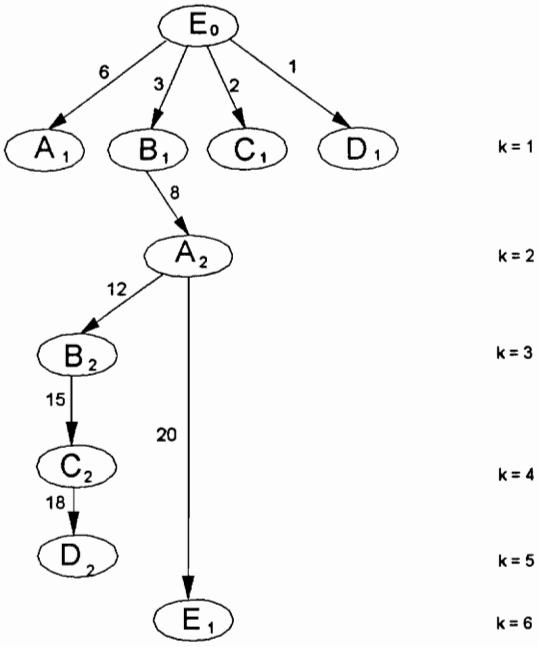


Figure 3.1 The Machining Sequence Tree of the Steel Plug

3.2 Identification of Tolerance Chains

Once the machining sequence tree of the part is generated, the second step is to find all the tolerance chains. The chains are needed for performing tolerance allocation, calculating operational (working) dimensions, and for verifying the process plan. There are two kinds of tolerance chains to be identified. The first class is the tolerance chains that affect the tolerance stackup of blueprint dimensions. The second class is the tolerance chains that affect the amount of stock removals at each operation. The following sections describe procedures to identify the tolerance chains of each class manually.

3.2.1 Tolerance Chains of Blueprint Dimensions

The working dimensions (or machining cuts) that contribute to the tolerance stackup of a blueprint dimension can be found from the machining sequence tree manually. For example, to find the tolerance chain of dimension DE of the steel plug, the final surfaces D and E must be first identified on the tree in Figure 3.2, they are the nodes labeled with D and E and having the largest subscripts, D_2 and E_1. All the arcs connecting these two nodes are the working dimensions that contribute to the tolerance stackup of this dimension. They are D_2C_2, C_2B_2, B_2A_2 and A_2E_1, which correspond to the machining operations 18, 15, 12, and 20. Similarly, the tolerance chains for the dimensions AE, BE, and CE of the steel plug can also be identified from the tree.

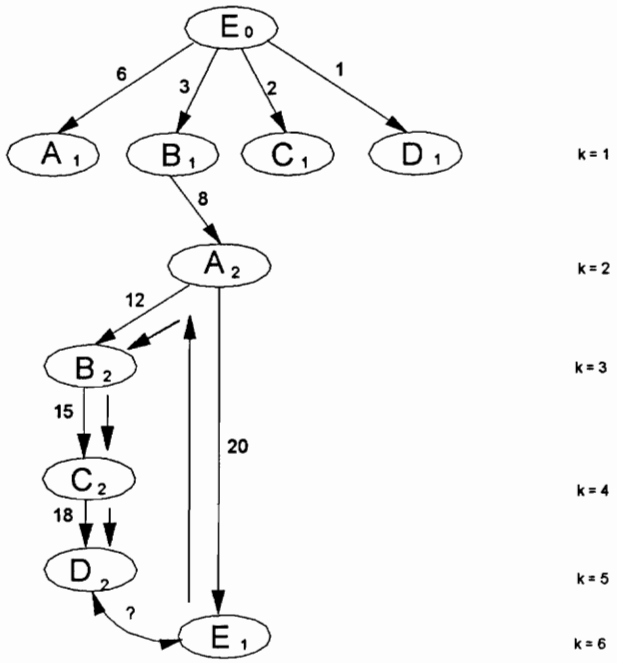


Figure 3.2 Tolerance Chain of a Dimension

3.2.2 Tolerance Chains of Stock Removals

To find the tolerance chains of stock removals manually, a similar procedure to the above one can be used. The amount of stock to be removed is the difference between two sizes of same surface, where one size represents before machining and the second corresponds to after machining. Therefore, the arcs connecting the two nodes with same letter and subscripts having a difference of 1 in the tree represent the tolerance chains of stock removals. For example, to determine the stock removal chain of stock removal number 18 in Figure 3.3, the node connected to the head of the arc 18, D_2, is one end of the tolerance chain. The other end of the tolerance chain must be a node with the same letter and having a subscript smaller by 1, D_1. All the arcs connecting D_1 and D_2 in Figure 3.3 contains all the working dimensions that affect the tolerance stackup of stock removal 18. These are D_2C_2, C_2B_2, B_2A_2, A_2B_1, B_1E_0, and E_0D_1, which correspond to the machining operations 18, 15, 12, 8, 3, and 1. Other tolerance chains of stock removals can be determined using this same procedure.

Unlike the traditional graphical representation method, the tree representation scheme is a convenient and direct method for capturing the relationships between the locating and the working surfaces, and the hierarchical structure exhibiting the evolution process of every surface explicitly. Therefore, the tolerance chain for resultant dimensions and stock removals can be recognized easily. To computerize the above manual procedures, an algorithm based on the tree representation was developed and is described in the succeeding section.

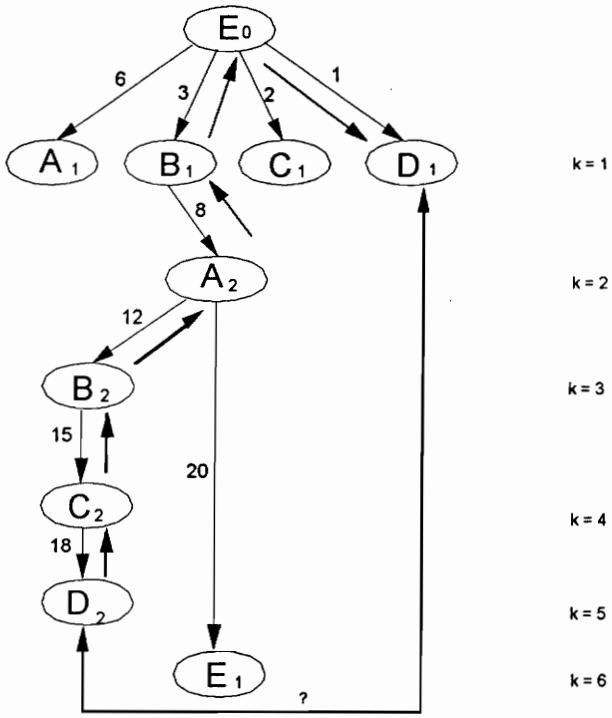


Figure 3.3 Tolerance Chain of a Stock Removal

3.3 A Computerized Algorithm to Find Tolerance Chains

To fully automate the process of tolerance charting, the tolerance chains must be identified automatially. To accomplish this, a computerized representation of the machining sequence tree is needed along with an algorithm to determine the tolerance chains. The predecessor index list and the level of nodes of the list structure can be used to computerize the tree representation [Bazaraa, et al., 1990]. It provides a one to one mapping of the tree components into a computerized data structure. The predecessor index list can be represented as follows:

$$p(i) = j \tag{3.1}$$

where i is a node number and j is the node number preceeding i on the chain from i to the root node (i = 1, ..., n; where n is the number of nodes in the tree). For the example in Figure 3.1, the predecessor index list is thus: $p(E_0) = 0$, $p(D_1) = E_0$, $p(C_1) = E_0$, $p(B_1) = E_0$, $p(A_1) = E_0$, $p(A_2) = B_1$, $p(B_2) = A_2$, $p(C_2) = B_2$, $p(D_2) = C_2$, $p(E_1) = A_2$.
The level of a node is defined as follow:

$$l(i) = C_i \tag{3.2}$$

where i is a node number and C_i is the number of arcs in the chain connecting node i to the root node. Therefore, in Figure 3.1, $l(E_1) = 0$, $l(D_1) = l(C_1) = l(B_1) = l(A_1) = 1$, $l(A_2) = 2$, $l(B_2) = l(E_1) = 3$, $l(C_2) = 4$, $l(D_2) = 5$.

An algorithm to find the tolerance chains automatically was developed with the following properties in mind [Foulds, 1992]:

1. Input: Specification of the information needed to begin,
2. Output: Specification of the results of an implementation,
3. Finiteness: Termination after a finite number of steps,
4. Definiteness: Absence of ambiguity, and
5. Effectiveness: Guarantee of the resolution of any problem instance.

The input information consists of the predecessor index list and the level of the node. The predecessor index list captures the relationships between nodes (or surfaces), and the level of nodes is used to identify the intersection node of the chain. The blueprint dimensions and stock removals are determined by two nodes in the machining tree. These nodes are identified manually from the tree as described in Section 3.2. The output is the chain of the nodes making up the tolerance chain of each blueprint dimension and stock removal. The following algorithm was developed to determine the tolerance chains of both blueprint dimensions and stock removals:

Step 1 Obtain the end nodes (i,j) of a tolerance chain from the input file and place in list L_n, namely, L_n = (i,j). Nodes in the tolerance chain will be added either after i or before j in the array of the list.

Step 2 If l(i) = l(j), then go to Step 4. If l(i) > l(j), trace the node i using the predecessor index list. Add the node p(i) after the node i in the list. If p(i) = j, the tolerance chain is found and consists of the nodes in the list L_n, and stop. Otherwise, go to next step.

Step 3 Repeat Step 2, but replace node i by the p(i).

Step 4 Trace the nodes i and j using the predecessor index list simultaneously. Add the node p(i) after the node i in the list, and add the p(j) before the node j in the list L_n. If p(j) = p(i), delete the repeat node (p(i) or p(j)) in the chain, and stop. Otherwise, go to the next step.

Step 5 Repeat Step 4, but replace nodes i and j by the p(i) and p(j).

For the steel plug example, the blueprint dimension 1.0 ± 0.020 in Figure 1.2 is delimited by surfaces D and E. Their node representations in the machining sequence tree (see Figure 3.1) are D_2 and E_1, respectively. The tolerance chain of this blueprint dimension will be found by the above algorithm as follows:

Step 1 Nodes (i, j) = (D_2, E_1). Set L_n = (D_2, E_1). l(D_2) = 5, l(E_1) = 3.

Step 2 Because l(D_2) > l(E_1), adding the node $p(D_2) = C_2$ in L_n. L_n = (D_2, C_2, E_1). Because of $(p(D_2) = C_2) \neq E_1$, go to next step.

Step 3 Because l(C_2) > l(E_1), adding the node $p(C_2) = B_2$ in L_n. L_n = (D_2, C_2, B_2, E_1). Because of $(p(C_2) = B_2) \neq E_1$, continue step 3.

Step 3 Because l(B_2) = l(E_1), go to Step 4.

Step 4 Add the node $p(B_2) = A_2$ in L_n. L_n = (D_2, C_2, B_2, A_2, E_1), and add the node $p(E_1) = A_2$ in L_n. L_n = (D_2, C_2, B_2, A_2, A_2, E_1). Since $p(B_2) = p(E_1)$, the nodes made of the tolerance chain of the dimension DE is found. Delete a repeat node A_2 in L_n, L_n = (D_2, C_2, B_2, A_2, E_1). Stop

Therefore, the tolerance chain of the blueprint dimension DE includes the nodes (D_2, C_2, B_2, A_2, E_1). The arcs of the tolerance chain are {(D_2, C_2), (C_2, B_2), (B_2, A_2), (A_2, E_1)}, which correspond to the machining operations 18, 15, 12, and 20, respectively.

3.4 An Optimization Model for Tolerance Allocation

Tolerance allocation is the process of determining the economical tolerance values for the working dimensions. The general principles for assigning the tolerances to the working dimensions are as follows:

1. The maximum possible tolerance should be assigned to each working dimension in the machining sequence as long as the design tolerance specifications are met.
2. Tolerance values assigned to each working dimension should be consistent with the designated process capability.
3. The chosen machining parameters such as the maximum and minimum stock removals for each operation should be consistent with the designated process capability.

The smaller the tolerance values, the higher the manufacturing costs. An ideal optimization model for tolerance allocation should therefore include manufacturing costs. However, these costs are usually not available, extremely difficult to estimate, and of the non-linear type (i.e., a cost value for a tolerance range). To overcome this problem, the developed optimization model utilizes weighted factors which indirectly represent

manufacturing costs, or other manufacturing attributes. These weighted factors are user specified since they depend on the available manufacturing facility. The value of each weighted factor will range between 0.1 and 1 to reflect the relative importance of a given operation in producing the part. The importance can be based on cost, time, quality, and/or flexibility [Chryssolouris, 1992]. The higher the value, the higher the importance. For example, a value of one would indicate to an expensive or high quality operation, and a value of 0.1 would indicate to a lengthy or highly flexible operation.

As indicated earlier in Chapters 1 and 2 any optimization model for tolerance allocation should treat the tolerance chains of resultant dimensions and stock removals separately. The tolerances of the working dimensions contributing to the tolerance stackup of a blueprint dimension should be relaxed as much as possible without exceeding the design tolerance; While the tolerances of the working dimensions contributing only to the tolerance stackup of a stock removal should be relaxed as much as possible without exceeding the processing capabilities of equipment. The developed optimization model consists of two modules: primary (blueprint dimension) operational tolerance allocation and secondary (stock removal) operational tolerance allocation. The following two sections explain these two modules and an illustrative example is provided in a section after that.

3.4.1 Primary Operational Tolerance Allocation

Before one formulates a linear mathematical model for the primary operational tolerance allocation, the decision variables and their constraints must be identified from

the problem. The working tolerances that affect the tolerances of blueprint dimensions directly are the decision variables. These variables are restricted by design specifications and process capabilities. To consider the characteristics of the different machining operation, such as machining cost, time, other manufacturing factors, the weighted factor for each operation is introduced in the objective function. Therefore, the generalized formulation of the primary tolerance allocation module can be stated as:

Maximize $$f_1 = \sum_{i \in D_p} w_i T_i \tag{3.3}$$

Subject to $$\sum_{i \in D_k} T_i \le b_k \qquad \text{for k = 1,...,o} \tag{3.4}$$

$$up_i \ge T_i \ge lp_i \tag{3.5}$$

Where:

- i — index of machining operations involved in primary tolerance allocation.
- T_i — tolerance to be allocated for the ith operation.
- w_i — the weight factor of the ith operation.
- k — index of blueprint dimensions.
- b_k — tolerance of the kth blueprint dimension.
- up_i — the upper economical limit of the tolerance for the ith operation.
- lp_i — the lower economical limit of the tolerance for the ith operation.
- Dp — all machining operations involved in the primary tolerance allocation
- D_k — machining operations made up of a dimension chain for the kth blueprint dimension.
- o — the number of blueprint dimensions.

The objective function of the primary tolerance allocation is to maximize the summation of the operational tolerances that directly impart the design tolerance specifications, in which the characteristics of each operation are taken into consideration using the weight factor. The first set of constraints (3.4) ensures that the accumulation of tolerances in the tolerance chain of each blueprint dimension is within the limit of the design tolerance specification. The second set of constraints (3.5) ensures that every operational tolerance to be assigned must be less than the upper economical tolerance of the designated process and greater than the lower economical tolerance of the specified process.

3.4.2 Secondary Operational Tolerance Allocation

The secondary operational tolerance allocation module deals with the tolerances of working dimensions that affect the tolerance stackup of stock removals. There are two factors to consider in this module. First, the tolerance of each stock removal must be less than amount of the stock to be removed to prevent costly scraps. Second, the amount of the stock to be removed should be within the upper and lower limits of the designated machine tool capability. These limits depend on the machine power, desired precision, and the surface finish requirements. Based on the above two factors, a linear programming model for the secondary operational tolerance allocation is formulated as follows:

Maximize $$f_2 = \sum_{j \in D_s} w_j T_j \tag{3.6}$$

Subject to $$\sum_{j \in D_r} T_j + S_r \le A_{\max} \quad \text{for r = 1, ..., q} \tag{3.7}$$

$$S_r - \sum_{j \in D_r} T_j \ge A_{\min} \quad \text{for r = 1, ..., q} \tag{3.8}$$

$$up_j \ge T_j \ge lp_j \tag{3.9}$$

where:

- j — index of machining operations involved in the secondary tolerance allocation.
- T_j — tolerance to be allocated for the jth operation.
- w_j — the weight factor of the jth operation.
- r — index of stock removal.
- q — the number of stock removals.
- Ds — all machining operations involved in the secondary tolerance allocation.
- D_r — machining operations made up of a dimension chain for the rth stock removal.
- S_r — the rth stock removal.
- up_j — the upper economical limit of the tolerance for the jth operation.
- lp_j — the lower economical limit of the tolerance for the jth operation.
- A_{max} — the permissible upper limit of the amount of stock removal of the processes.
- A_{min} — the permissible lower limit of the amount of stock removal of the processes.

The objective function of the secondary operational tolerance allocation is similar to that of the primary operational tolerance allocation, which is to maximize the summation of tolerances of working dimensions related to stock removals. However, the constraints are different. The first set of constraints (3.7) ensures that the upper limit of stock to be removed at a given machine is satisfied. The second set of constraints (3.8) ensures that the amount of stock to be removed is larger than its tolerance accumulation by at least the amount of variability inherent in the process. The third set of constraints (3.9) ensures that the tolerance of each working dimension is less than the upper economical tolerance of the designated process and greater than the lower economical tolerance of the specified process.

3.4.3 An Illustrative Example

According to the above optimization model of the primary tolerance allocation, we must first determine the decision variables (T_i), the working tolerances that directly affect the tolerances of blueprint dimensions. In the steel plug example, there are four blueprint dimensions (i.e., four tolerance chains in the machining sequence tree). The design tolerances of these blueprint dimensions will be the right-hand-side terms of constraints (3.4), which are 0.005, 0.002, 0.009, and 0.020, respectively. The working tolerances in each tolerance chain are the components of the left-hand-side terms of constraints (3.4). By assuming same weight factor for all operations (i.e., $w_i = 1.0$), the linear mathematical model can be formulated as

Maximize $T_{20} + T_{18} + T_{15} + T_{12}$ (3.10)

Subject to $T_{20} \leq 0.005$ (3.11)

$T_{20} + T_{12} \leq 0.002$ (3.12)

$T_{20} + T_{15} + T_{12} \leq 0.009$ (3.13)

$T_{20} + T_{18} + T_{15} + T_{12} \leq 0.020$ (3.14)

$0.012 \geq T_{20}, T_{18}, T_{15}, T_{12} \geq 0.0005$ (3.15)

In this mathematical model, T_{20}, T_{18}, T_{15} and T_{12} are four working (operational) tolerances to be determined. They directly affect the design tolerance specifications of the part. The constraints (3.11)-(3.14) ensures that the operational tolerances in the four tolerance chains are less than or equal to the design tolerance specifications. The last set of constraints ensure that the operational tolerances should be less than or equal to 0.015, the upper limit of the economical tolerance of equipment, and greater than or equal to 0.0005, the lower limit of the economical tolerance of equipment. The results from this model are the operational tolerances of four working dimensions. The remaining operational tolerances will be determined based on the second optimization model.

Likewise, there are five stock removals during the process of manufacturing the part according to the process planning. They yield five tolerance chains in the machining sequence tree. The working tolerances in each tolerance chain determine the left-hand-side terms of constraints (3.7) and (3.8). The right-hand-side terms of these constraints are A_{max} and A_{min}, which are determined by the processing capabilities of the designated machines. For simplification, a weight factor of 1.0 is assigned to all operations. The linear mathematical model can be formulated as:

Maximize $T_8 + T_6 + T_3 + T_2 + T_1$ (3.16)

Subject to $T_8 + T_6 + T_3 + S_8 \le A_{\max}$ (3.17)

$T_{12} + T_8 + S_{12} \le A_{\max}$ (3.18)

$T_{15} + T_{12} + T_8 + T_3 + T_2 + S_{15} \le A_{\max}$ (3.19)

$T_{18} + T_{15} + T_{12} + T_8 + T_3 + T_1 + S_{18} \le A_{\max}$ (3.20)

$T_{20} + T_8 + T_6 + S_{20} \le A_{\max}$ (3.21)

$S_8 - (T_8 + T_6 + T_3) \ge A_{\min}$ (3.22)

$S_{12} - (T_{12} + T_8) \ge A_{\min}$ (3.23)

$S_{15} - (T_{15} + T_{12} + T_8 + T_3 + T_2) \ge A_{\min}$ (3.24)

$S_{18} - (T_{18} + T_{15} + T_{12} + T_8 + T_3 + T_1) \ge A_{\min}$ (3.25)

$S_{20} - (T_{20} + T_8 + T_3) \ge A_{\min}$ (3.26)

$0.020 \ge T_1, T_2, T_3, T_6, T_8 \ge 0.001$ (3.27)

Since T_{20}, T_{18}, T_{15} and T_{12} are known from the primary tolerance allocation model, their values can be directly substituted into the secondary tolerance allocation model. The remaining T_1, T_2, T_3, T_6, and T_8 are now the decision variables. The constraints (3.17)-(3.21) ensure that tolerance accumulation in the tolerance chains plus the amount of stock to be removed, which is the worst case, must not exceed the permissible upper limit of the amount of stock removal of the processes ($A_{\max}$). The constraints (3.22)-(3.26) ensure that the amount of stock to be removed less the tolerance accumulation in each tolerance chain is greater than or equal to the permissible lower limit of the amount of stock removal of the processes ($A_{\min}$). The set of constraints (3.27) ensure that the operational tolerances are less than or equal to 0.010, the upper limit of the economical tolerance of equipment, and greater than or equal to 0.001, the lower limit of the economical tolerance of equipment. In the constraint equations, S_8, S_{12}, S_{15}, S_{18}, and S_{20} are 0.02, 0.008,

0.017, 0.032, and 0.01, respectively. They are obtained from the tentative process plan, and usually based on choosing practical values appropriate to the machining operations being performed. Once the maximum (A_{max}) and minimum (A_{min}) machining allowances are given, optimal solutions for all the remaining operational tolerances can be obtained by solving the model. The solution of this problem is given in Section 4.4.

3.5 Working Dimensions and Tolerances of Stock Removals

The last task to be performed to complete the tolerance chart is the determination of the working dimensions and the tolerances of the stock removals. The following two sections describe the methods developed for calculating them.

3.5.1 Working Dimensions

The working dimensions can be determined by solving a set of linear equations. Generally, if there are N nodes in the machining sequence tree, there are N-1 arcs, which are equal to the number of operations or working dimensions. We can find exact N-1 linear equations from the machining sequence tree, which can be proven as follows. Assuming that the part has M dimensions. It must have M+1 surfaces, which correspond to M+1 nodes in the machining sequence tree. Based on M dimensions of the part, they yield M tolerance chains or linear equations. Each extra operation will add a new node and a new arc in the tree (i.e., a surface is machined twice). This will yield a new tolerance chain or a new linear equation. Therefore, if the part has M dimensions, it needs

M1 stock removals except the solid stock removals, the following mathematical expression holds true: M+M1 = N-1. These equations can be constructed based on the dimension tolerance chains and stock removal tolerance chains. Each tolerance chain yields a linear equation. The working dimensions can be determined by solving such a set of linear equations simultaneously. These equations can be denoted as follows:

$$[C]_{n\times n}[D]_{n\times 1} = [S]_{n\times 1} \tag{3.28}$$

where:

n the number of operations (arcs) in the machining sequence tree.

D $[d_1, d_2, \ldots d_n]^T$ is a vector of the unknown working dimensions.

S $[s_1, s_2, \ldots s_n]^T$ is a vector of the known stock removals and blueprint dimensions.

c_{ij} = 1 if d_j is a positive component of s_i.

-1 if d_j is a negative component of s_i.

0 otherwise.

To determine c_{ij} of the matrix C, it is equivalent to determining the signs of the working dimensions in the tolerance chains of stock removals and blueprint dimensions, since the elements in the matrix C can take either 1, -1, or 0. If working dimensions are calculated by hand, it is quite simple to determine the signs of the working dimensions in the tolerance chain. However, to computerize the calculation process, a method for distinguishing between the signs of the working dimensions in the tolerance chains must be developed.

Prior to finding the rules, the characteristics of stock removals at every surface of a part are analyzed. There are two different nature types of surfaces in the part when the surface is distinguished by its external normal vector if the part is laid down horizontally. One type of the surface is leftward; while the other type of the surface is rightward. They can be distinguished by the following rule. When the part is laid down horizontally, a point at a surface in question is taken. If the external normal vector of the surface drawn at that point is leftward, then the surface direction is leftward. If the external normal of the surface drawn at that point is rightward, then the surface direction is rightward. When the surface directions are combined with the tolerance chains identified from the machining sequence tree, the procedure for determining the signs of working dimensions is developed. To address the procedure clearly, the previous node label of the machining sequence tree will be replaced by the number, the node labeled with the letter A is replaced with 1, the node B is replaced by 2, and so on. The converted tree is shown in Figure 3.4. The tolerance chains of five stock removals (SR8, SR12, SR15, SR18, and SR20) are discovered from the tree as shown in Figure 3.5. Since the arrow of each arc does not matter for determining the signs of the working dimensions, it is omitted in the figures.

Each tolerance chain has two end nodes. For a stock removal tolerance chain, it means that a certain amount of material is cut off at the surface of the end node. If the surface of the end node is leftward, the end node of the tolerance chain with the larger subscript is designated as the right end, and denoted by R. The other end is the left end, and denoted by L as shown in Figure 3.6. If the surface of the end node is rightward, the end node of the tolerance chain with the larger subscript is designated as the left end (L).

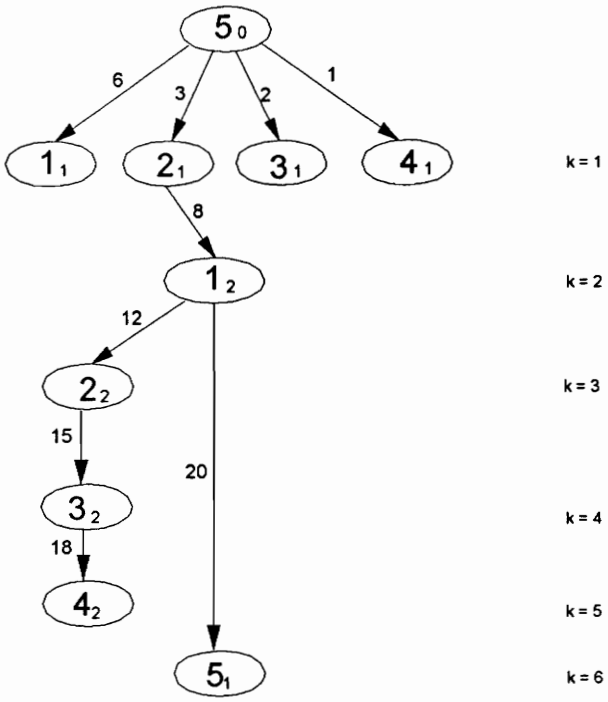


Figure 3.4 A Machining Sequence Tree with Numbered Nodes

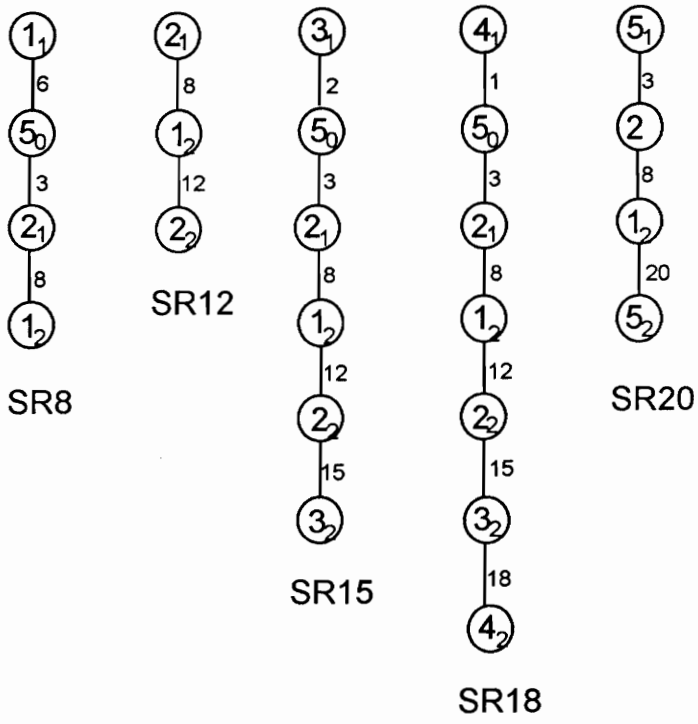


Figure 3.5 Tolerance Chains of Stock Removals

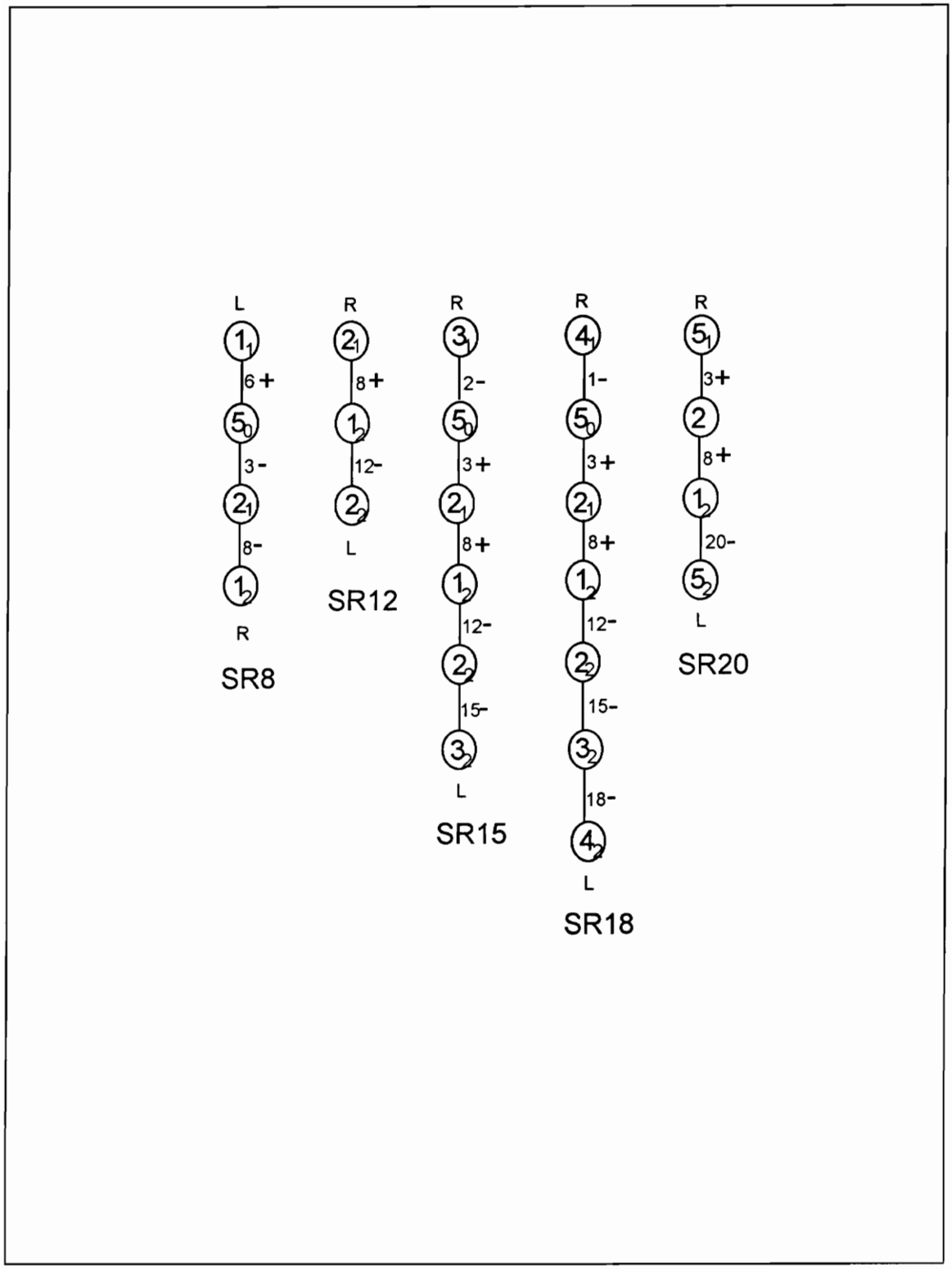
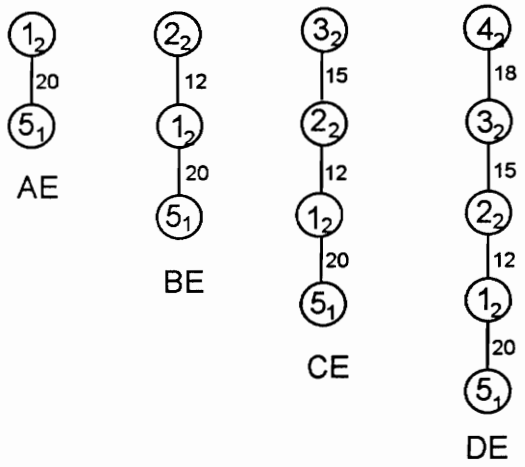


Figure 3.6 Signed Working Dimensions in the Stock Removal Tolerance Chain

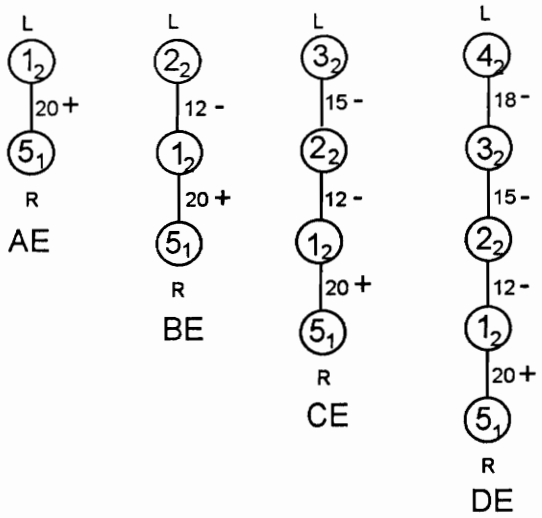
The other end is the right end (R). Once the directions of tolerance chains are determined, the procedure for distinguishing the signs of working dimensions is described below.

Assuming that we start from the right end of the tolerance chain. If the number of the right node of the tolerance chain is greater than the number of the left node, then the operation between the adjacent nodes belongs to the decreasing group, or the working dimension takes "-" sign at the calculation of the stock removal; otherwise, the operation belongs to the positive group, or the working dimension takes "+" sign. If we start from the left end of the tolerance chain, the consequence is opposite. Namely, if the number of the left node of the certain operation of the tolerance chain is smaller than the number of the right node, than this operation between the adjacent nodes belongs to the decreasing group; otherwise, the operation belongs to the positive group. Based on the above rules, the working dimension (or operation) signs for each tolerance chain of stock removals are determined as illustrated in Figure 3.6.

To determine the working dimension signs of the dimensional tolerance chain, it is similar to the above method, but instead of using the surface direction to determine the direction of the tolerance chain, the direction of the dimension tolerance chain is determined by the number of end nodes. The end node that has the larger number than the other end nodes is the right end node (R), and the other node is the left end node (L), since we label the surfaces of the part from left to right in an ascending order using the number 1, 2, 3, and so on. When the direction of the tolerance chain is decided, the sign of each operation is determined by the same rules as the tolerance chain of the stock removal. The tolerance chains of four design dimensions (AE, BE, CE, and DE) identified from the machining sequence tree are shown in Figure 3.7a, and the working dimension



(a) Dimensional tolerance chains



(b) Signed Working dimension

Figure 3.7 Dimension Tolerance Chains and Signed Working Dimension

signs determined by the above method are shown in Figure 3.7b. Hence, the coefficient matrix of working dimensions of the used example determined by the above rules is as follow:

$$\begin{bmatrix} 0 & 0 & -1 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 1 & -1 & -1 & 0 & 0 \\ -1 & 0 & 1 & 0 & 1 & -1 & -1 & -1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & 1 \end{bmatrix}$$

3.5.2 Tolerances of Stock Removals

The tolerance of each stock removal is the allowable variation from the mean stock removal. Once the tolerance chain of each stock removal is identified and the working tolerance of each operation is determined, the tolerance of each stock removal is the sum of the tolerances of all the involved working dimensions. For example, the tolerance of the stock removal at the surface 4 or D (see Figure 3.6) is equal to the sum of working tolerances of operations 1, 3, 8, 12, 15, and 18.

CHAPTER 4

COMPUTER IMPLEMENTATION

This chapter describes a computerized optimization system for tolerance charting based on the developed methodology in Chapter 3. The program was developed in the programming language C. A user's manual is given in Appendix A, and a program listing is included in Appendix B. Section 4.1 discusses the format of the input file and how it can be generated. Section 4.2 explains the integration and implementation procedures of the developed methodology into a software. Section 4.3 discusses the program output and the results obtained from the steel plug example. Section 4.4 discusses the results of three different methods. Section 4.5 describes the application of the software on another example.

4.1 Input File

The main function of the tolerance chart is to check the feasibility of the process plan and to determine the machining parameters of each operation, working dimensions and their tolerances. The main input requirements for constructing the tolerance chart are a tentative process plan and the processing capabilities of the designated equipment. The developed manual procedure for constructing the machining sequence tree directly from the tentative process plan, described in Section 3.1, can be used to generate the tree. To computerize the tree structure, the predecessor index list which can be stored in an array is

used to represent the relationship between nodes. However, this representation of trees does not contain all the information required to generate the tolerance chart automatically. Hence, instead of using an one-dimensional array to represent the machining sequence tree on the computer, a structure containing all required information is used. This is one of the main advantages of the programming language C, which permits grouping several variables of different types into a single logical entity. The structure is specified by the node, and stores all the information extracted from the machining sequence tree and other data, such as the surface direction and the node direction of a dimension. The predecessor index list is only one part of this structure.

An example of an input file is shown in Figure 4.1. The first row has three pieces of information, number of nodes in the tree, number of blueprint dimensions and number of stock removals. Starting from the second row, each node occupies one row consisting of thirteen pieces of information. From left to right, they are node number, operation number, precedence node, corresponding stock removal node, corresponding dimension node, dimension tolerance, amount of stock removal, blueprint dimension, lower limit of the process tolerance, upper limit of the process tolerance, surface direction, node direction at that dimension, and surface order number. The first five columns are directly extracted from the machining sequence tree. The first column is the node number or the identifier of the node. The second column is the operation number or the attribute of the arc between the node and its precedence node in the tree. The third column is the precedence node or the parent node of the corresponding node in the first column. The fourth and fifth columns are the corresponding nodes of stock removals and blueprint dimensions. For the fourth column, if the stock removal at that node is solid, the entry is zero. Solid stock removal means that the surface is first machined at that operation. For

```
10 4 5
1   0  0 0 0 0.0000 0.0000 0.0  0.0000  0.0000   1  0 5
2   1   1 0 0 0.0000 0.0000 0.0  0.0030  0.0100  1  0 4
3   2   1 0 0 0.0000 0.0000 0.0  0.0030  0.0100  1  0 3
4   3   1 0 0 0.0000 0.0000 0.0  0.0030  0.0100  1  0 2
5   6   1 0 0 0.0000 0.0000 0.0  0.0030  0.0100 -1  0 1
6   8   4 5 0 0.0000 0.0200 0.0  0.0040  0.0100 -1  0 1
7   20 6 1 6 0.0050 0.0110 4.0  0.0010  0.0100  1  1 5
8   12 6 4 7 0.0020 0.0080 3.0  0.0010  0.0100  1 -1 2
9   15 8 3 7 0.0090 0.0170 2.0  0.0030  0.0100  1 -1 3
10 18 9 2 7 0.0200 0.0320 1.0  0.0150  0.0200  1 -1 4
```

□

Figure 4.1 Example of an Input File

the fifth column, if the nodes are initial nodes or intermediate nodes, the entry is zero. Note that only the final nodes on the machining sequence tree have the corresponding dimensional node. The six and seven columns record the dimension tolerance and the amount of the stock removal occurred at that node. The eighth and ninth columns record the upper and lower limit of the process tolerance for the designated operation. The last three columns contain the information for determining the signs of working dimensions automatically, and they are the surface direction, the node direction of the dimension, and the order number of the surface. The surface direction possesses two directions: leftward and rightward. The leftward surface is denoted by -1 at the tenth column, the rightward surface is denoted by 1. Likewise, a blueprint dimension has a left endpoint and a right endpoint; which correspond to the left node and the right node in the tolerance chain. The left node is denoted by -1; while the right node is denoted by 1. The order number of the surface simply replaces the letter label of the part surface, which is assigned from left to right in an ascending order for the computer processing.

4.2 Program Structure

A flow diagram of the program is shown in Figure 4.2. The input is a data file that must be generated according to the format described in the previous section. The program reads this file by the function *input_file* and stores them into the corresponding elements of the structure. The intermediate procedures include finding the tolerance chains of both blueprint dimensions and stock removals, optimization of operational tolerance allocation, and calculation of working dimensions and tolerances of stock removals. All these procedures are described in detail in the subsequent sections. The

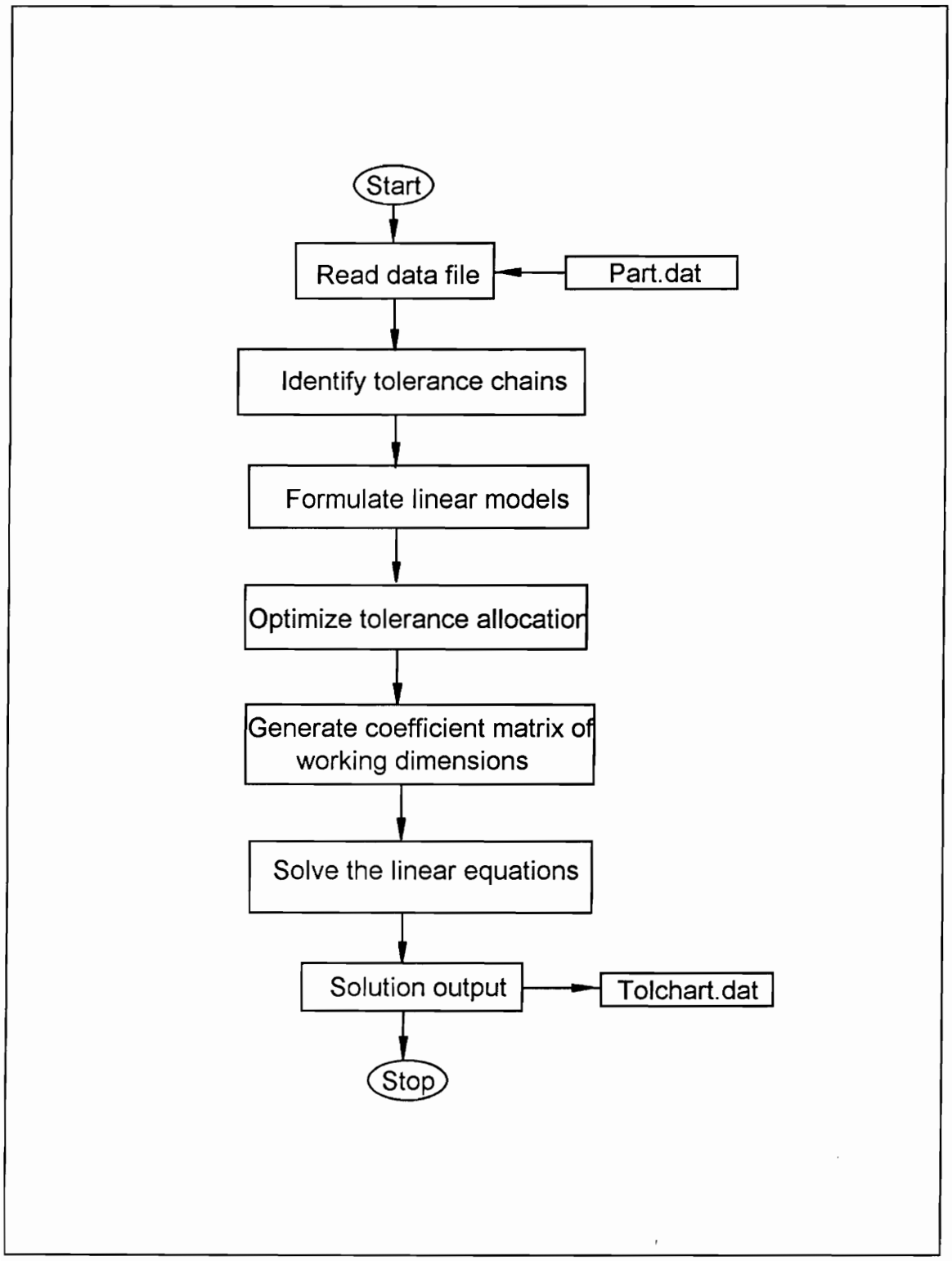


Figure 4.2 Flow Diagram of the Tolerance Control Software

program output is a tolerance chart stored in the filename TOLCHART.DAT, which contains the optimal machining parameters of each operation, and is generated by the function *save_file*.

Identification of the tolerance chains: The identification of tolerance chains is divided into two groups: blueprint dimension tolerance chains and stock removal tolerance chains. The algorithm for identifying the tolerance chains has been discussed in Section 3.3. They are coded in two functions *D_tolerance_chain* and *ST_tolerance_chain*.

Optimization of tolerance allocation: The optimization model for tolerance allocation has been described in Sections 3.4.1 and 3.4.2. The objective function is to maximize the operational tolerances. The decision variables for the primary tolerance allocation problem are the working tolerances that directly affect the design tolerances. Once the tolerance chains for the blueprint dimensions of the part are identified by the function *D_tolerance_chain*, the decision variables can be determined, all the working tolerances involved in producing those blueprint dimensions.

The constraints of the optimization model have two distinct sets: (1) design tolerance specifications, and the processing capabilities of the designated equipment. Each design tolerance specification constraint is constructed using the working tolerances in the tolerance chain for each blueprint dimension. The process capabilities are retrieved from the input file. The model's coefficients are determined by the function *primary_datafile*, and then stored into the filename PT.DAT for access by subsequent functions to solve the optimization model. To explain the process, a simple example is given below:

Maximize $z = 4x_1 + 6x_2 + 12x_3 + 3x_4$ (4.1)

Subject to $x_1 + x_2 + x_3 + x_4 \leq 0.020$ (4.2)

$x_1 + x_2 + x_4 \leq 0.009$ (4.3)

$x_1 + x_4 \leq 0.002$ (4.4)

$x_4 \leq 0.005$ (4.5)

$x_1, x_2, x_3, x_4 \geq 0.0005$ (4.6)

By introducing slack variables y_i and artificial variables z_i, constraints can be written in the following format:

$z_1 = 0.020 - x_1 - x_2 - x_3 - x_4 - y_1$ (4.7)

$z_2 = 0.009 - x_1 - x_2 - x_4 - y_2$ (4.8)

$z_3 = 0.002 - x_1 - x_4 - y_3$ (4.9)

$z_4 = 0.005 - x_4 - y_4$ (4.10)

$z_5 = 0.0005 - x_1 + y_5$ (4.11)

$z_6 = 0.0005 - x_2 + y_6$ (4.12)

$z_7 = 0.0005 - x_3 + y_7$ (4.13)

$z_8 = 0.0005 - x_4 + y_8$ (4.14)

Since the mathematical model is a linear optimization problem, the simplex method is used to find the optimal solution. A tableau format is used to store the information. Therefore, the above problem can be summarized in Table 4.1. The table entries inside the box of double lines are the coefficients of the original problem organized into a tabular format. These entries, along with the values of the number of decision variables (N), the number of constraints (M), the number of constraints that are less than or equal to b_i

($(b_i \geq 0)$ (m1), the number of constraints that are greater than or equal to b_i ($(b_i \geq 0)$ (m2), and the number of constraints that are equal to b_i ($(b_i \geq 0)$ (m3), are the only input that is required by the simplex method. This is the reason that the function *primary_datafile* only generates the coefficients of the model based on the solutions of identifying the tolerance chains and input information.

To implement the simplex method in the computer program, the two-phase method is used to solve the problem. The first phase produces an initial feasible basic vector, using an auxiliary objective function. The second phase solves the problem produced by the first phase, using the original objective function. The simplex method is coded in the function *solving_primary_tol*. For the secondary tolerance allocation module, the function *secondary_datafile* is used to generate the data file, and the linear programming model is solved by the function *solving_secondary_tol*.

Calculation of working dimensions: The working dimensions are calculated by a set of linear equations, which are generated from the dimension tolerance chains and stock removal tolerance chains. Each tolerance chain yields a linear equation. The number of equations is equal to the number of unknown working dimensions. Hence, the working dimensions can be uniquely determined by such a set of linear equations. To computerize this process, the key issue is to determine the coefficient matrix, since a resultant dimension is simple addition or subtraction of the involved working dimensions. The elements of the coefficient matrix are composed of 1, -1, and 0. The procedures for determining these elements was addressed in Section 3.5.1, and can be summarized into two steps: (1) identifying the direction of surface (or node direction of the dimension), retrieved from the input file; and (2) comparing the two nodes adjacent to each other in

Table 4.1 Tableau Format of the Linear Programming

		x_1	x_2	x_3	x_4	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8
z	0	4	6	12	3	0	0	0	0	0	0	0	0
z_1	0.020	-1	-1	-1	-1	-1	0	0	0	0	0	0	0
z_2	0.009	-1	-1	0	-1	0	-1	0	0	0	0	0	0
z_3	0.002	-1	0	0	-1	0	0	-1	0	0	0	0	0
z_4	0.005	0	0	0	-1	0	0	0	-1	0	0	0	0
z_5	0.0005	-1	0	0	0	0	0	0	0	1	0	0	0
z_6	0.0005	0	-1	0	0	0	0	0	0	0	1	0	0
z_7	0.0005	0	0	-1	0	0	0	0	0	0	0	1	0
z_8	0.0005	0	0	0	-1	0	0	0	0	0	0	0	1

the tolerance chain to decide on the signs of the working dimension. These procedures are coded in the function *coefficient_matrix*, which generates all the elements of the coefficient matrix. Once the coefficient matrix is determined, the working dimensions are solved by the Gaussian elimination method, which is coded in the function *solving_equation*.

4.3 Program Output

An example of the program output for the results of the steel plug example in Section 1.2 is given in Figure 4.3. Note that the weighted factors of all the working tolerances in both objective functions are assigned 1.0. The output file records machining parameters of each operation, design specifications, and the final results. The first column is the operation number. The second and third columns are the working dimensions and their tolerances. The fourth and fifth columns are the stock removals and their tolerances at each operation. Below the double dashed lines, the blueprint dimensions and their tolerances are listed on the left side, and the resultants of the tolerance chart are listed on the right side. If the process plan is feasible, the tolerances of resultants should be equal to or less than the tolerances of blueprint dimensions. The ideal condition is that the corresponding tolerances of blueprint dimensions and resultants in the tolerance chart are equal.

Table 4.2 provides a summary of the results obtained by the manual tolerance chart method (Figure 1.3) and the developed one (Figure 4.3). The optimization method relaxed three operational tolerances, T_1, T_2, and T_6, more than the manual method.

TOLERANCE CHART

OPRT NO	WORK DIM	TOLERANCE	STOCK REM	TOLERANCE
1	0.9790	0.0050	SOLID	0.0000
2	1.9940	0.0050	SOLID	0.0000
3	3.0030	0.0030	SOLID	0.0000
6	4.0310	0.0120	SOLID	0.0000
8	1.0080	0.0040	0.0200	0.0190
12	1.0000	0.0010	0.0080	0.0050
15	1.0000	0.0030	0.0170	0.0160
18	1.0000	0.0150	0.0320	0.0310
20	4.0000	0.0010	0.0110	0.0080

BLUEPRINT		RESULTANT	
DIMENSION	TOLERANCE	DIMENSION	TOLERANCE
4.0000	0.0050	4.0000	0.0010
3.0000	0.0020	3.0000	0.0020
2.0000	0.0090	2.0000	0.0050
1.0000	0.0200	1.0000	0.0200

Figure 4.3 Output of the Program

Table 4.2 Summary Results of the Optimization and Manual Methods

Method	T_1	T_2	T_3	T_6	T_8	T_{12}	T_{15}	T_{18}	T_{20}	ΣT
Optim	.005	.005	.003	.012	.004	.001	.003	.015	.001	.049
Manual	.003	.003	.003	.010	.004	.001	.003	.015	.001	.043

Otherwise stated, higher manufacturing cost reduction can be achieved.

4.4 Comparison with an existing linear model

The steel plug example was also used by Irani et al. [1989] to check the performance of their linear model. However, it is not clear why they changed the initial parameters, amount of stock removals and processing capabilities, of the example [Wade, 1967, 1983]. The results obtained by their linear model was compared with the manual method's results (see Table 4.3). Three of the operational tolerances are tighter than those obtained by the manual method. To provide a valid comparison with their model, the same input parameters from their paper were used. Figure 4.4 shows the input file of their data, and Figure 4.5 provides the results obtained by the developed model. Table 4.3 provides a summary of the results obtained from all three methods. It should be noted that the input data of the manual method is different than those used by the optimization methods. The tolerance values of working dimensions affecting only stock removals are T_1 to T_8, while T_{12} to T_{20} are those affecting the blueprint dimensions. The results indicate that the two models obtained the same tolerance values for the working dimensions affecting the blueprint dimensions, T_{12}, T_{15}, T_{18}, and T_{20}, while for those working dimensions affecting only the stock removals there are some disagreements. Three of the operational tolerances (T_1, T_2, and T_6) are greater than their results, while the last two (T_3 and T_8) are tighter than their results. However, by comparing the sum of all the operational tolerances, it is clear that the developed optimization model outperformed their model. The sum of the tolerances is relaxed by roughly 18% (i.e., better cost savings). It should be noted that most of the relaxation of tolerances were in

```
10 4  5
1   0   0  0 0 0.0000 0.0000 0.0  0.0000  0.0000  1  0 5
2   1   1  0 0 0.0000 0.0000 0.0  0.0010  0.0200  1  0 4
3   2   1  0 0 0.0000 0.0000 0.0  0.0010  0.0200  1  0 3
4   3   1  0 0 0.0000 0.0000 0.0  0.0010  0.0200  1  0 2
5   6   1  0 0 0.0000 0.0000 0.0  0.0010  0.0200 -1  0 1
6   8   4  5 0 0.0000 0.0200 0.0  0.0010  0.0120 -1  0 1
7   20 6  1 6 0.0050 0.0100 4.0  0.0005  0.0120  1  1 5
8   12 6  4 7 0.0020 0.0060 3.0  0.0005  0.0120  1 -1 2
9   15 8  3 7 0.0090 0.0175 2.0  0.0005  0.0120  1 -1 3
10 18 9  2 7 0.0200 0.0320 1.0  0.0005  0.0120  1 -1 4
□
```

Figure 4.4 Changed Input file of the Steel Plug

TOLERANCE CHART

OPRT NO	WORK DIM	TOLERANCE	STOCK REM	TOLERANCE
1	0.9780	0.0105	SOLID	0.0000
2	1.9925	0.0070	SOLID	0.0000
3	3.0040	0.0010	SOLID	0.0000
6	4.0300	0.0170	SOLID	0.0000
8	1.0060	0.0010	0.0200	0.0190
12	1.0000	0.0005	0.0060	0.0015
15	1.0000	0.0070	0.0175	0.0165
18	1.0000	0.0110	0.0320	0.0310
20	4.0000	0.0015	0.0100	0.0035

BLUEPRINT		RESULTANT	
DIMENSION	TOLERANCE	DIMENSION	TOLERANCE
4.0000	0.0050	4.0000	0.0015
3.0000	0.0020	3.0000	0.0020
2.0000	0.0090	2.0000	0.0090
1.0000	0.0200	1.0000	0.0200

Figure 4.5 Tolerance Chart of the Steel Plug at the condition of the Initial Parameters Used by Iran et al. [1989]

Table 4.3 Results of Three Methods

Method	T_1	T_2	T_3	T_6	T_8	T_{12}	T_{15}	T_{18}	T_{20}	ΣT
1	.0105	.0070	.0010	.0170	.0010	.0005	.0070	.0110	.0015	.0565
2	.0050	.0015	.0030	.0115	.0055	.0005	.0070	.0110	.0015	.0465
3	.0030	.0030	.0030	.0100	.0040	.0010	0.003	.0150	.0010	.0430

1 Developed optimization model

2 Irani et al.'s linear model

3 Manual tolerance chart method

the working dimensions affecting only the stock removals (T_1 to T_8). This supports the importance of optimizing the tolerances of the working dimensions into two separate groups.

Compared with Irani et al.'s linear model, the developed optimization model not only takes all the processing capability constraints into consideration, but also optimizes the tolerances of working dimensions into two separate groups. Also, weighted factors are provided in the objective functions to allow users to set optimization priorities based on cost or other manufacturing attributes, such as time, quality, and/or flexibility. Although Irani et al.'s model indirectly utilizes weighted factors, function of the frequency of a working dimension appearing in different tolerance chains, these factors are highly influenced by the manual tolerance chart method. The higher the frequency, the tighter the tolerance. However, the frequency is not the only factor that govern the tolerance allocation process and is highly dependent on the tentative process plan. Nonetheless, their weights were included in the developed optimization model and the results are shown in Table 4.4. The developed optimization model still provided higher sum of tolerances than their model. An improvement of at least 6% is achieved. Again, this can be contributed to the fact that the developed optimization model optimizes the tolerances of working dimensions into two separate groups of tolerance chains, blueprint dimensions and stock removals.

Table 4.4 Results of Two Optimization Methods

Method	T_1	T_2	T_3	T_6	T_8	T_{12}	T_{15}	T_{18}	T_{20}	ΣT
weights	1	1	4	1	5	6	4	2	5	
1	.0010	.0085	.0010	.0125	.0065	.0015	.0070	.0110	.0005	.0495
2	.0050	.0015	.0030	.0115	.0055	.0005	.0070	.0110	.0015	.0465

1 Developed optimization model

2 Irani et al.'s linear model

4.4 A Case Study

To further validate the effectiveness of the developed methodology, another example, adopted from Eary and Johnson [1962], was utilized. The blueprint drawing of the part is illustrated in Figure 4.6. The machining sequence tree generated based on the process plan is shown in Figure 4.7. Note that only the machining operations are included in the machining sequence tree. From the machining sequence tree, the first five columns of the input file of the part can be determined. The nodes F_0, A_1, F_1, B_1, C_1, D_1, A_2, F_2, and E_1 in Figure 4.7 are denoted by 1, 2, 3, 4, 5, 6, 7, 8, and 9, respectively. The first column is the number of nodes, and the second one is the operation number or the attribute of the arc of the node at the fist column and its precedence node at the third column. Note that only the head nodes of the arcs, or the machining surfaces, have an operation number. For this reason node F_0 or 1, the initial location node, does not have an operation number, instead a zero is used. The third column is the precedence node of the first column, and it can be directly obtained from the tree. The fourth column is the stock removal node of the first column. The part has three stock removals, and hence there should be exactly three nonzero elements in this column. Each nonzero element of this column is one end node of the stock removal chain, the other end node is the corresponding node of the same row at the first column. The fifth column is the dimension node. For this part, five nonzero entries corresponding to the five blueprint dimensions should be present. From the sixth to the tenth columns, the entries are determined from the blueprint requirements and processing capabilities. The sixth column is the design tolerance, defined by the two nodes on the same row at the dimension node column and the first column. Following the same principle, the amount of the stock

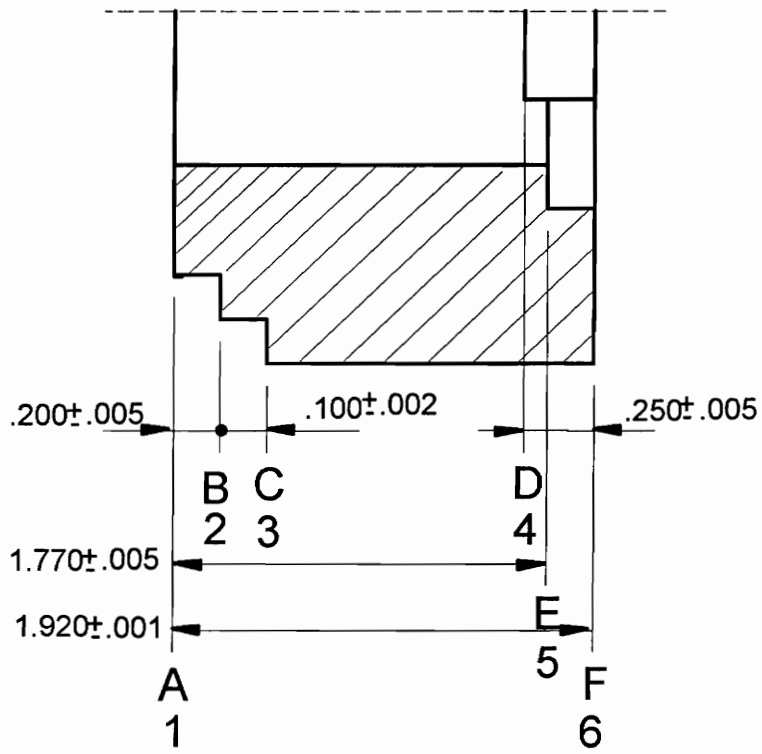


Figure 4.6 Blueprint Drawing of a Part

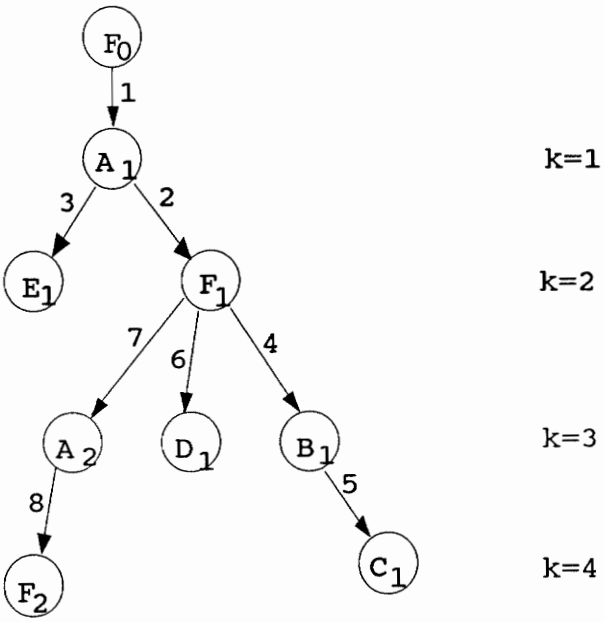


Figure 4.7 Machining Sequence Tree of the Part

removal and the blueprint dimension are stored in the seventh and eighth columns, respectively. The ninth and tenth columns record the lower tolerance and the upper tolerance of the process capability that corresponds to the operation number on the same row at the second column. The last three columns store the information for determining the signs of working dimensions in the tolerance chain. The eleventh column records the surface direction of the part. If the surface is leftward, it is denoted by -1, otherwise, 1. The twelfth column is the dimensional node direction at the first column. If the node at the first column is the left node of that dimension, then -1 is placed on the same row at the twelve column, otherwise, 1. The last column is the order number of the part surface, it is assigned from left to right in an ascending order. Figure 4.8 shows the generated input file of the part. In this case, the weighted factors of the working tolerances in the primary objective function are assigned two values: the weighted factor of the working tolerances takes 1.0, if the frequency of the working tolerances in the tolerances chains are larger than 1; otherwise 0.1. The weighted factors of the working tolerances in the secondary objective functions are assigned 1.0.

Figure 4.9 shows the final results of the part tolerance chart. The results of the optimization method and manual tolerance chart method are summarized in Table 4.5. The results of the optimization method are much better than those manually developed. Five operational tolerances, T_1, T_4, T_6, T_7 and T_8, are relaxed when comparing with before adjustment results. When comparing with the adjusted tolerance chart, there are still two operations tolerances being relaxed more, operations T_1 and T_6.

```
9 5 3
1  0  0 0 0  0.0000 0.0000 0.00 0.0000  0.0000  1  0 6
2  1  1 0 0  0.0000 0.0300 0.00 0.0020  0.0100 -1  0 1
3  2  2 1 0  0.0000 0.0300 0.00 0.0020  0.0100  1  0 6
4  4  3 0 0  0.0000 0.0000 0.00 0.0020  0.0100 -1  0 2
5  5  4 0 4  0.0020 0.0000 0.10 0.0020  0.0100 -1  1 3
6  6  3 0 8  0.0050 0.2600 0.25 0.0020  0.0100  1 -1 4
7  7  3 2 4  0.0050 0.0100 0.20 0.0005  0.0100 -1 -1 1
8  8  7 3 7  0.0010 0.0100 1.92 0.0005  0.0100  1  1 6
9  3  2 0 7  0.0050 0.0160 1.77 0.0020  0.0100  1  1 5
```

Figure 4.8 Input File of the Case Problem

TOLERANCE CHART

OPRT NO	WORK DIM	TOLERANCE	STOCK REM	TOLERANCE
1	1.9700	0.0030	0.0300	0.0000
2	1.9400	0.0020	0.0300	0.0050
3	1.7800	0.0020	0.0160	0.0000
4	1.7300	0.0040	SOLID	0.0000
5	0.1000	0.0020	SOLID	0.0000
6	0.2600	0.0030	0.2600	0.0000
7	1.9300	0.0010	0.0100	0.0030
8	1.9200	0.0010	0.0100	0.0020

BLUEPRINT		RESULTANT	
DIMENSION	TOLERANCE	DIMENSION	TOLERANCE
0.1000	0.0020	0.1000	0.0020
0.2500	0.0050	0.2500	0.0050
0.2000	0.0050	0.2000	0.0050
1.9200	0.0010	1.9200	0.0010
1.7700	0.0050	1.7700	0.0050

Figure 4.9 Tolerance Chart of the Part

Table 4.5 Results of the Optimization and Manual Methods for the Case Problem

Method	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	ΣT
Optim.	0.010	0.002	0.002	0.004	0.002	0.003	0.001	0.001	0.025
Bef. adj.	0.002	0.002	0.002	0.002	0.002	0.002	.0005	.0005	0.013
Aft. adj.	0.002	0.002	0.002	0.004	0.002	0.002	0.001	0.001	0.016

CHAPTER 5

CONCLUSIONS

This thesis has presented a systematic optimization method and algorithms to automate the tolerance chart method. The developed methods include a representation scheme for capturing the machining sequence of a part, an algorithm for identifying the tolerance chains, an optimization model for tolerance allocation, and a procedure for calculating the working dimensions. Based on these systematic procedures, a stand alone computer package was developed using the C programming language to integrate and automate all these procedures in a unified software for tolerance charting. Two examples, commonly found in the literature, were utilized to verify and validate the developed algorithms and the computer package. Also, the performance of the developed algorithms was compared to another published linear model for tolerance allocation. The major conclusions are summarized below.

The tree representation is the most direct and effective method as compared to other representation schemes, such as a matrix or a graph, for capturing the machining sequence. Because the tree has a unique chain connecting each pair of its nodes, the tolerance chains of blueprint dimensions and stock removals are uniquely defined. All other information required for constructing the tolerance chart can also be captured in the tree structure. The tree is computerizable through the predecessor index list. Based on this representation of the machining sequence, an algorithm was developed to determine the tolerance chains of both blueprint dimensions and stock removals. The algorithm is

simple and suitable for computer implementation as compared with other algorithms found in the literature.

To ideally minimize manufacturing costs, the working dimensions should be optimized in two separate groups, based on their direct effect on blueprint dimensions and stock removals. The working dimensions affecting only the tolerances of stock removals can be relaxed more than those directly affecting the chains of the resultant or design dimensions. For this reason, the developed optimization model consisted of two modules: primary and secondary operational tolerance allocation. The primary module deals with the tolerances of working dimensions affecting blueprint dimensions, and the secondary module optimizes the remaining tolerances of working dimensions. The model also included all the necessary constraints as compared to other optimization models in the literature. Finally, weight factors are included in the model so that the user can set them to reflect manufacturing costs indirectly or any other priority, such as time or the type of cut (rough or finish).

For the calculation of working dimensions, a set of linear equations was constructed from the tolerances chains. This eliminates the need for the difficult backtracking process of the manual tolerance chart method. The working dimensions are calculated by applying the Gaussian elimination technique on the linear equations. This procedure ensures the uniqueness of the working dimensions, since the equations are derived from the tolerance chains. Additionally, a method was developed to determine the signs of the working dimensions in the tolerance chains.

Finally, a computer program was developed to integrate the identification of tolerance chains, optimization of the tolerance allocation, and calculation of the working dimensions together. It is a stand alone software and can run on IBM compatible microcomputers. The input to the program is a data file that must be generated from the machining sequence tree and the process plan according to the designated format. The final result of the program is the tolerance chart of the part.

From the results obtained from two experimental examples, the computerized optimization model not only avoids the tedious and time-consuming manual tolerance chart method, but also gives better solutions. The developed model was also compared against another model in the literature. The results obtained from the developed model are around 18% better than those obtained by the other model.

Possible future works include the following:

1. The weighted factors can be replaced by a step type cost function to better reflect actual manufacturing costs.
2. Equipment selection: If several types of equipment are available to perform a given operation, the model can be expanded to perform optimization over all available equipment.
3. Integration with a CAD software: The software can be integrated with a CAD software to provide the necessary graphics.
4. Integration with a CAPP software: The software can be integrated with a computer-aided process planning system to verify the tentative process plan and/or determine the optimal machining parameters.

REFERENCES

Ahluwalia, R.S. and Karolin, A.V., 1984, "CATC - A Computer Aided Tolerance Control System," *Journal of Manufacturing Systems*, Volume 3, No. 2.

Bazaraa, M. S., Jarvis, J., and Sherali, H. D., 1990, *Linear Programming and Network Flows*, John Wiley & Sons, New York.

Chang, Tien-Chien, Wysk, Richard A., and Wang, Hsu-Pin, 1991, *Computer-Aided Manufacturing*, Prentice Hall, Englewood Cliffs, New Jersey.

Chase, K.W., and Greenwood, W.H., 1988, "Design Issues in Mechanical Tolerance Analysis," *Manufacturing review*, Vol. 1, No.1, pp. 50-59.

Chryssolouris, Geoge, 1992, *Manufacturing Systems: Theory and Practice*, Springer-Verlag.

Eary, D. F., and Johnson, G. E., 1962, *Process Engineering for Manufacturing*, Prentice Hall, Englewood Cliffs, New Jersey.

Hoffman, P., 1982, "Analysis of Tolerances and Process Inaccuracies in Discret Part Manufacturing," *Computer Aided Design*, Volume 14, Number 2.

Fainguelernt, D., Weill, R., and Bourdet, P., 1986, "Computer Aided Tolerancing and Dimensioning in Process Planning," *Annals of the CIRP*, Volume 35, 1.

Farmer, L.E. and Harris, A.G., 1984, "Change of Datum of the Dimensions on Engineering Design Drawings," *Int. J. Mach. Tool Des. Res.* Vol. 24, No. 4, pp. 267-275.

Foulds, L.R., 1992, *Graph Theory Applications*, Springer-Verlag.

Irani, S.A., Mittal, R.O., and Lehtihet, E.T., 1989, "Tolerance Chart Optimization," *Int. J. Prod.*, Vol. 27, No. 9, 1131-1552.

Ji, P., 1993, "A Tree Approach for Tolerance Charting," *Int. J. Prod.*, Vol. 31, No. 5, 1023-1033.

Karolin, A.V., 1984, "Computer Aided Tolerance Analysis", *AUTOFACT 5 conference*.

Li, K.J., and Zhang, C., 1989, "Operational Dimensions and Tolerances Calculation in CAPP Systems for Precision manufacturing," *Annals of the CIRP*, Vol. 38, 1.

Ostwald, P.F., and Huang, J., 1977, "A Method for Optimal Tolerance Selection," *Journal of Engineering for Industry*.

Press, W. H., Flannery, B. P., Teukolsky, S. A., and Verterling, W. T., 1988, *Numerical Recipes in C: The Art of Scientific Computing,* Cambridge University Press.

Roy, U., Liu, C.R. and Woo, T.C., 1991, "Review of Dimensioning and Tolerancing: Representation and Processing," *Computer-Aided Design*, Volume 23, Number 7.

Schrage, L., 1989, *User's Manual for Linear, Integer and Quadratic programming with LINDO,* South San Francisco.

Tipins, Vijay A., 1988, "Process and Economic Models for Manufacturing Operations," *Design and Analysis of Integrated Manufacturing Systems*, edited by W.Dale Compton, pp. 92-117.

Xiaoqing, Tang, and Davies, B.J., 1988, "Computer Aided Dimensional Planning," *Int. J. Prod.*, Vol. 26, No. 2, 283-297.

Weill, R., 1988, "Integrating Dimensioning and Tolerancing in Computer-Aided Process Planning," *Robotics & Computer-Integrated Manufacturing*, Volume 4, Number 1//2.

Wade, O.R., 1967, *Tolerance Control in Design and Manufacturing*, Industrial Press Inc., New York.

Wade, O.R., 1983, "Tolerance Control," *Tool and Manufacturing Engineers Handbook*, Vol. 1, Machining, edited by T.J. Drozda and C. Wick, pp.2-1-2-60.

Whybrew, K., Britton, G.A., Robinson, D.F., and Sermsuti-anuwat, Y., 1990, "A Graph-Theoretic Approach to Tolerance Charting," *Int. J. Adv. Manuf. Technol.*, 5:175-183.

Zhang, H.C., and Huq, M.E., 1992, "Tolerancing Techniques: the state-of-the-art," *Int. J. Prod.*, Vol. 30, No. 9, 2111-2135.

APPENDIX A

USER'S MANUAL

TO

TOLCHART

VERSION 1.1

Software for Generating the Tolerance chart
for the IBM PC and Compatibles

Introduction: The program implements an optimization method for generating the tolerance chart of a part. The program contains three parts: file input, intermediate procedures, and output. The input file must be generated according to the format specified in this thesis. Two example input files are shown in Figure 4.1 and 4.8. The intermediate procedures realize the various procedures of tolerance charting. The output is the tolerance chart and is stored in the filename TOLCHART.DAT, if the given process plan is feasible. Otherwise, the program will immediately cease and give error information. This version allows for a maximum of 40 nodes, 20 blueprint dimensions, and 20 stock removals.

Requirements: In order to run this version, TOLCHART requires an IBM PC or compatible running under PC-DOS or MS-DOS version 2.0 (or higher).

Installation: The only file required for using the software is TOLCHART.EXE. Installation on a hard disk is accomplished by copying that file to the hard disk.

Running TOLCHART: This program is a stand alone package. It only requires the user to input the data filename at the prompt when running the program. The followings are procedures to operate the program.

1. After starting the machine and loading DOS, you may execute TOLCHART by placing the TOLCHART system diskette in the default drive and typing TOLCHART. If you have a hard disk, you will want to copy the file TOLCHART.EXE to a directory on your disk. You may then run TOLCHART directly from your hard disk.

2. The program will ask for the filename at the prompt ENTER YOUR FILENAME. There are two sample files, test1.dat and test2.dat included in the TOLCHART diskette.

3. After you input the filename of the input data, it will take a while for the computer to make computations and output the final results, this is dependent upon the computer you use. The final results are both displayed on the screen for user's convenience and stored in the filename TOLCHART.DAT in the driver you use automatically.

APPENDIX B

Program Listing

```
#include <stdio.h>
#include <math.h>

#define ROW                     40          /* the maximum number of constriants */
#define COLUMN                  40          /* the maximum number of variables */
#define MAX_NODE                40          /* the maximum number of nodes */
#define MAX_DIM                 20          /* the maximum number of dimensions */
#define MAX_ST                  20          /* the maximum number of stock removal */
#define M_NUM_OF_CHAIN          20          /* the maximum number of nodes in a chain */
#define MAX_STOCK_ALLOWANCE     0.08
#define MIN_STOCK_ALLOWANCE     0.001
#define EPS 1.0e-6

/*************************************************************************
**
** PROTOTYPES
**
*/
void inputfile(void);
void D_tolerance_chain(void);
void ST_tolerance_chain(void);
void calculating_FO_D(void);
void calculating_FO_ST(void);
void primary_datafile(void);
void solving_primary_tol(void);
void solving_secondary_tol(void);
void stock_removal_datafile(void);
void right_hand_coeff(void);
void right_hand_coeff_nim(void);
void beep(void);
void simplx(float **a, int m, int n, int m1, int m2, int m3,
                int *icase, int izrov[], int iposv[]);
void simp1(float **a, int mm, int ll[], int nll, int iabf, int *kp,
                float *bmax);
void simp2(float **a, int n, int l2[], int n12, int *ip, int kp, float *q1);
void simp3(float **a, int i1, int k1, int ip, int kp);
int *ivector(long nl, long nh);
float **convert_matrix(float *a, long nrl, long nrh, long ncl, long nch);
void stock_removal_tol(void);
void coefficient_matrix(void);
void solving_equation(void);
void save_output(void);
void display_output(void);
```

```
void calculating_design_tol(void);


struct node_info {
        int node;
        int oprt;
        int pre_node;
        int stock_node;
        int dim_node;
        float dtol;
        float stock_removal;
        float dim;
        float process_tol;
        float up_process_tol;
        int s_direction;
        int d_direction;
        int s_order;
        float stock_tol;
        float wdim;
        float tol;
};

struct node_info p[MAX_NODE];
int listcn[MAX_DIM][M_NUM_OF_CHAIN];
int st_chain[MAX_ST][M_NUM_OF_CHAIN];
int F_O_D[MAX_NODE+1], F_O_ST[MAX_NODE+1];
float tol[MAX_DIM], st_tol[MAX_ST], b[ROW];
float c[MAX_NODE][MAX_NODE], d[MAX_NODE];
int NODE_NUM, NUMBER_OF_D, NUMBER_OF_ST;

/*****************************************************************************
**
** MAIN PROGRAM AND FUNCTIONS
**
*/
main ()
{
struct node_info *data_ptr;

        data_ptr = (struct node_info *)
                        calloc(MAX_NODE, sizeof(struct node_info));
        if( data_ptr == NULL) {
          printf("\nAllocation memery failure.");
          exit(1);
        }
        else {
          inputfile();
          D_tolerance_chain();
          ST_tolerance_chain();
          calculating_FO_D();
          calculating_FO_ST();
```

```
            primary_datafile();
            solving_primary_tol();
            stock_removal_datafile();
            solving_secondary_tol();
            stock_removal_tol();
            coefficient_matrix();
            solving_equation();
           calculating_design_tol();
            save_output();
            display_output();
          }
          free(data_ptr);
}

/* ************************************************************************
**
** FUNCTION inputfile
** Read the relevant data from the file name "mst.dat"
**
*/
void inputfile(void)
{
FILE *file_ptr;
int i;
char filename[20];

          clrscr();
          printf("\nENTER FILENAME: ");
          gets(filename);

          file_ptr = fopen(filename, "r");
          if(file_ptr == NULL)  {
           printf("\n Cannot open the file  %s.", filename);
           exit(1);
          }
          else {
           fscanf(file_ptr, "%d %d %d\n", &NODE_NUM, &NUMBER_OF_D,
                                          &NUMBER_OF_ST);
           for(i=0; i<=NODE_NUM; i++) {
            fscanf(file_ptr, "%d %d %d %d %d %f %f %f %f %f %d %d %d\n",
                   &p[i].node, &p[i].oprt, &p[i].pre_node, &p[i].stock_node,
                   &p[i].dim_node, &p[i].dtol, &p[i].stock_removal, &p[i].dim,
                   &p[i].process_tol, &p[i].up_process_tol, &p[i].s_direction,
                   &p[i].d_direction, &p[i].s_order);
           }
          }
          fclose(file_ptr);
}

/* ************************************************************************
**
```

```
** FUNCTION D_tolerance_chain
** Identify the tolerance chain of blueprint dimensions.
**
*/
void D_tolerance_chain(void)
{
int i, j, k1, k2, m, m1, n1, n2, temp;
int temp_list[10];

        i = 0;
        for(m1=1; m1<=NODE_NUM; m1++) {
         if(p[m1-1].dim_node != 0 && p[m1-1].dim_node < p[m1-1].node) {
          k1 = p[m1-1].node-1;
          k2 = p[m1-1].dim_node-1;

          for(j=0; j<10; j++)  /* initializing the array */
           listcn[i][j] = 0;

          listcn[i][0] = p[k1].node;
          j = 1;   /* finding all the precedence nodes of one end of */
          while(p[k1].pre_node != 0 ) {          /* tolerance chain */
           if(p[k1].pre_node != p[k2].node) {
            listcn[i][j] = p[k1].pre_node;
            j++;
           }
           else {
            break;   /* finding the tolerance chain */
           }
           k1 = p[k1].pre_node - 1;
          }

          for(n1=0; n1<10; n1++)   /* initializing the array */
           temp_list[n1] = 0;

          /* finding all the precedence nodes of the other node */
          temp_list[0] = p[k2].node;
          m = 1;

          while(p[k2].pre_node != 0 && p[k1+1].node != p[k2].node) {
           temp_list[m] = p[k2].pre_node;
           m++;
           k2 = p[k2].pre_node - 1;
          }

          n1 = 0;
          while(n1<j) {
           n2 = 0;
           while(n2<m)  {
            if(listcn[i][n1] == temp_list[n2]) {
                  temp = n1;
                  while(n2 != 0)  {
```

```
            listcn[i][temp] = temp_list[n2-1];
            n2--;
            temp++;
          }
           for(; temp <10; temp++)
             listcn[i][temp] = 0;
         n2 = m;
         n1 = j;
        }
        n2++;
 }
 n1++;
 }
 i++;
}
if(p[m1-1].dim_node != 0 && p[m1-1].dim_node > p[m1-1].node) {
 k1 = p[m1-1].dim_node-1; /* p[m1-1].node-1;*/
 k2 = p[m1-1].node-1; /* p[m1-1].dim_node-1;*/

 for(j=0; j<10; j++)  /* initializing the array */
  listcn[i][j] = 0;

 listcn[i][0] = p[k1].node;
 j = 1;    /* finding all the precedence nodes of one end of */
 while(p[k1].pre_node != 0 ) {          /* tolerance chain */
  if(p[k1].pre_node != p[k2].node) {
   listcn[i][j] = p[k1].pre_node;
   j++;
  }
  else {
   break;   /* finding the tolerance chain */
  }
  k1 = p[k1].pre_node - 1;
 }

 for(n1=0; n1<10; n1++)   /* initializing the array */
  temp_list[n1] = 0;

 /* finding all the precedence nodes of the other node */
 temp_list[0] = p[k2].node;
 m = 1;

 while(p[k2].pre_node != 0 && p[k1+1].node != p[k2].node) {
  temp_list[m] = p[k2].pre_node;
  m++;
  k2 = p[k2].pre_node - 1;
 }

 n1 = 0;
 while(n1<j) {
  n2 = 0;
```

```
            while(n2<m) {
              if(listcn[i][n1] == temp_list[n2]) {
                    temp = n1;
                    while(n2 != 0) {
                      listcn[i][temp] = temp_list[n2-1];
                      n2--;
                      temp++;
                    }
                     for(; temp <10; temp++)
                       listcn[i][temp] = 0;
                    n2 = m;
                    n1 = j;
                  }
                  n2++;
             }
             n1++;
            }
            i++;
          }
         }
}

/* ************************************************************************
**
** FUNCTION ST_tolerance_chain
** Identify the tolerance chain of stock removals.
**
*/
void ST_tolerance_chain(void)
{
int i, j, k1, k2, m, m1, n1, n2, temp;
int temp_list[10];

          i = 0;
          for(m1=1; m1<=NODE_NUM; m1++) {
           if(p[m1-1].stock_node != 0 && p[m1-1].stock_node < p[m1-1].node) {
            k1 = p[m1-1].node-1; /* st[i][0]-1 */
            k2 = p[m1-1].stock_node-1; /* st[i][1]-1 */

            for(j=0; j<10; j++)  /* initializing the array */
              st_chain[i][j] = 0;

            st_chain[i][0] = p[k1].node;
            j = 1;      /* finding all the precedence nodes of one node */
            while(p[k1].pre_node != 0 ) {
             if(p[k1].pre_node != p[k2].node) {
              st_chain[i][j] = p[k1].pre_node;
              j++;
             }
             else {
              break;   /* finding the tolerance chain */
```

```
  }
  k1 = p[k1].pre_node - 1;
 }

 for(n1=0; n1<10; n1++)   /* initializing the array */
  temp_list[n1] = 0;
 /* finding all the precedence nodes of the other node */
 temp_list[0] = p[k2].node;
 m = 1;
 while(p[k2].pre_node != 0) {
  temp_list[m] = p[k2].pre_node;
  m++;
  k2 = p[k2].pre_node - 1;
 }

 n1 = 0;
 while(n1<j) {
  n2 = 0;
  while(n2<m)  {
   if(st_chain[i][n1] == temp_list[n2]) {
         temp = n1;
         while(n2 != 0)  {
           st_chain[i][temp] = temp_list[n2-1];
           n2--;
           temp++;
         }
          for(; temp <10; temp++)
            st_chain[i][temp] = 0;
         n2 = m;
         n1 = j;
        }
        n2++;
  }
  n1++;
 }
 i++;
}

if(p[m1-1].stock_node != 0 && p[m1-1].stock_node > p[m1-1].node)  {
 k1 = p[m1-1].stock_node-1; /* p[m1-1].node-1; */
 k2 = p[m1-1].node-1; /* p[m1-1].stock_node-1;*/

 for(j=0; j<10; j++)  /* initializing the array */
  st_chain[i][j] = 0;

 st_chain[i][0] = p[k1].node;
 j = 1;     /* finding all the precedence nodes of one node */
 while(p[k1].pre_node != 0 ) {
  if(p[k1].pre_node != p[k2].node) {
   st_chain[i][j] = p[k1].pre_node;
   j++;
```

```
            }
            else {
              break;   /* finding the tolerance chain */
            }
            k1 = p[k1].pre_node - 1;
           }

           for(n1=0; n1<10; n1++)   /* initializing the array */
             temp_list[n1] = 0;
           /* finding all the precedence nodes of the other node */
           temp_list[0] = p[k2].node;
           m = 1;
           while(p[k2].pre_node != 0) {
             temp_list[m] = p[k2].pre_node;
             m++;
             k2 = p[k2].pre_node - 1;
           }

           n1 = 0;
           while(n1<j) {
             n2 = 0;
             while(n2<m)  {
               if(st_chain[i][n1] == temp_list[n2]) {
                     temp = n1;
                     while(n2 != 0)  {
                       st_chain[i][temp] = temp_list[n2-1];
                       n2--;
                       temp++;
                     }
                      for(; temp <10; temp++)
                        st_chain[i][temp] = 0;
                     n2 = m;
                     n1 = j;
                    }
                    n2++;
             }
             n1++;
           }
           i++;
         }
        }
}

/*****************************************************************************
**
** FUNCTION: Calculating_FO_D
** CALLS:
** RETURNS:
**
*/
void calculating_FO_D(void)
```

```
{
int i, j, k;

        k = 0;
        while(k<=NODE_NUM) {
          F_O_D[k] = 0;
          for(i=0; i<NUMBER_OF_D; i++)
            for(j=0; j<10; j++)
              if(listcn[i][j] == k && k !=0)
                  F_O_D[k] += 1;
          k++;
        }

}


/*****************************************************************************
**
** FUNCTION: Calculating_FO_OF_ST
** CALLS:
** RETURNS:
**
*/

void calculating_FO_ST(void)
{
int i, j, k;

        k = 0;
        while(k<=NODE_NUM) {
          F_O_ST[k] = 0;
          for(i=0; i<NUMBER_OF_ST; i++)
            for(j=0; j<10; j++)
              if(st_chain[i][j] == k && k != 0)
                  F_O_ST[k] += 1;
          k++;
        }
}

/* ***************************************************************************
**
** FUNCTION primary_datafile
** Generate the datafile of the primary tolerance allocation.
**
*/
void primary_datafile(void)
{
FILE *out_file;
int i, j, k, l, n, m, m1, m2, m3;

        n = 0;
```

```
    for(i=0; i<=NODE_NUM; i++)
      if(F_O_D[i] != 0)
        n += 1;

    m  = NUMBER_OF_D + 2*n;
    m1 = NUMBER_OF_D + n;
    m2 = n;
    m3 = 0;

    out_file = fopen("tolp.dat", "w");
    if( out_file == NULL) {
     printf("\n ph.dat can't open.");
     exit(2);
    }
    else
     fprintf(out_file, "%d %d %d %d %d\n", m, n, m1, m2, m3);

    /* objective function */
    fprintf(out_file, "%10.5f", 0.0);
    for(j=0; j<=NODE_NUM; j++) {
      if(F_O_D[j] !=0 && F_O_D[j] > 1)
        fprintf(out_file, "%10.5f", 1.0);
      if(F_O_D[j] !=0 && F_O_D[j] ==1)
        fprintf(out_file, "%10.5f", 0.1);
    }
    fprintf(out_file, "\n");

    i = 0;
    for(l=1; l<=NODE_NUM; l++) {
     if(p[l-1].dtol != 0.0)  {
       fprintf(out_file, "%10.5f", p[l-1].dtol);

       for(j=0; j<=NODE_NUM; j++) {
              if(F_O_D[j] != 0 ) {
                for(k=0; k<10; k++) {
                  if(listcn[i][k] == j) {
                        fprintf(out_file, "%10.5f", -1.0);
                        k = 19;
                  }
                }
                if(k != 20)
                  fprintf(out_file, "%10.5f", 0.0);
              }
       }
       fprintf(out_file, "\n");
       i++;
     }
    }

/* process capability constraints */
```

```
        k=0;
        i=0;
        for(j=0; j<=NODE_NUM; j++) {
          if(F_O_D[j] != 0 ) {
            fprintf(out_file, "%10.5f", p[j-1].up_process_tol);
                for(k=0; k<n; k++) {
                  if( k == i)
                    fprintf(out_file, "%10.5f", -1.0);
                  else
                    fprintf(out_file, "%10.5f", 0.0);
                }
                fprintf(out_file, "\n");
                i++;
          }
        }

        k=0;
        i=0;
        for(j=0; j<=NODE_NUM; j++) {
          if(F_O_D[j] != 0 ) {
            fprintf(out_file, "%10.5f", p[j-1].process_tol);
                for(k=0; k<n; k++) {
                  if( k == i)
                    fprintf(out_file, "%10.5f", -1.0);
                  else
                    fprintf(out_file, "%10.5f", 0.0);
                }
                fprintf(out_file, "\n");
                i++;
          }
        }

        fclose(out_file);
}

/* ************************************************************************
**
** FUNCTION stock_removal_datafile
** Generate the datafile of the secondary tolerance allocation.
**
*/
void stock_removal_datafile(void)
{
FILE *out_file;
int i, j, k, n, m, m1, m2, m3;

        n = 0;
        for(i=0; i<=NODE_NUM; i++)
          if(F_O_ST[i] != 0 && F_O_D[i] == 0)
            n += 1;
```

```
m  = 2*NUMBER_OF_ST + 2*n;
m1 = 2*NUMBER_OF_ST + n;
m2 = n;
m3 = 0;

out_file = fopen("tols.dat", "w");
if( out_file == NULL) {
  printf("\n stock tolerance datafile can't open.");
  exit(2);
}
else
  fprintf(out_file, "%d %d %d %d %d\n", m, n, m1, m2, m3);

fprintf(out_file, "%10.5f", 0.0);/* objective function coefficient */
for(j=0; j<=NODE_NUM; j++)  {
   if(F_O_ST[j] !=0 && F_O_D[j] == 0 && F_O_ST[j] > 1)
     fprintf(out_file, "%10.5f", 1.0);
   if(F_O_ST[j] !=0 && F_O_D[j] == 0 && F_O_ST[j] == 1)
     fprintf(out_file, "%10.5f", 1.0);
}
fprintf(out_file, "\n");

right_hand_coeff();

i = 0;  /* constraints */
while( i<NUMBER_OF_ST )  {
   fprintf(out_file, "%10.5f", st_tol[i]);
   for(j=0; j<=NODE_NUM; j++) {
     if(F_O_ST[j] != 0 && F_O_D[j] == 0)  {
           for(k=0; k<10; k++) {
             if(j == st_chain[i][k]) {
               fprintf(out_file, "%10.5f", -1.0);
               k =19;
             }
           }
           if(k != 20)
             fprintf(out_file, "%10.5f", 0.0);
     }
   }
   fprintf(out_file, "\n");
   i++;
}

right_hand_coeff_nim();

i = 0;  /* constraints */
while( i<NUMBER_OF_ST )  {
   fprintf(out_file, "%10.5f", st_tol[i]);
   for(j=0; j<=NODE_NUM; j++) {
     if(F_O_ST[j] != 0 && F_O_D[j] == 0)  {
           for(k=0; k<10; k++) {
```

```
                if(j == st_chain[i][k])  {
                  fprintf(out_file, "%10.5f", -1.0);
                  k = 19;
                 }
               }
               if(k != 20)
                fprintf(out_file, "%10.5f", 0.0);
          }
        }
        fprintf(out_file, "\n");
        i++;
      }

      /* process capability constraints */

      k=0;
      i=0;
      for(j=0; j<=NODE_NUM; j++) {
        if(F_O_ST[j] != 0 && F_O_D[j] == 0 ) {
          fprintf(out_file, "%10.5f", p[j-1].up_process_tol);
               for(k=0; k<n; k++) {
                 if( k == i)
                   fprintf(out_file, "%10.5f", -1.0);
                 else
                   fprintf(out_file, "%10.5f", 0.0);
               }
               fprintf(out_file, "\n");
               i++;
        }
      }

      k=0;
      i=0;
      for(j=0; j<=NODE_NUM; j++) {
        if(F_O_ST[j] != 0 && F_O_D[j] == 0 ) {
          fprintf(out_file, "%10.5f", p[j-1].process_tol);
               for(k=0; k<n; k++) {
                 if( k == i)
                   fprintf(out_file, "%10.5f", -1.0);
                 else
                   fprintf(out_file, "%10.5f", 0.0);
               }
               fprintf(out_file, "\n");
               i++;
        }
      }

      fclose(out_file);
}

/* ************************************************************************
```

```
**
** FUNCTION right_hand_coeff
** Calculate the right-hand-side coefficients of constraints.
**
*/
void right_hand_coeff(void)
{
int i, j, k, m, n;
float sum,temp_st[10];

        k = 0;
        for(j=0; j<NODE_NUM; j++) {
          if(p[j].stock_removal != 0.0 ) {
            temp_st[k] = p[j].stock_removal;
            k++;
          }
        }

        sum = 0.0;
        m = 0;
        n = 0;
        for(j=0; j<NUMBER_OF_ST; j++) {
          while(st_chain[m][n] != 0 ) {
            for(i=0; i<=NODE_NUM; i++) {
              if( F_O_D[i] != 0 && i == st_chain[m][n]) {
                    sum += p[i-1].tol;
                    i = NODE_NUM;
              }
            }
            n++;
          }
          st_tol[j] = MAX_STOCK_ALLOWANCE - sum - temp_st[j];
          sum = 0.0;
          n = 0;
          m++;
        }
}

/* ************************************************************************
**
** FUNCTION right_hand_coeff_nim
** Calculate the right-hand-side coefficients of constraints.
**
*/
void right_hand_coeff_nim(void)
{
int i, j, k, m, n;
float sum,temp_st[10];

        k = 0;
        for(j=0; j<NODE_NUM; j++) {
```

```
            if(p[j].stock_removal != 0.0 ) {
              temp_st[k] = p[j].stock_removal;
              k++;
            }
          }

          sum = 0.0;
          m = 0;
          n = 0;
          for(j=0; j<NUMBER_OF_ST; j++) {
            while(st_chain[m][n] != 0)  {
              for(i=0; i<=NODE_NUM; i++) {
                if( F_O_D[i] != 0 && i == st_chain[m][n]) {
                      sum += p[i-1].tol;
                      i = NODE_NUM;
                }
              }
              n++;
            }
            st_tol[j] = temp_st[j] - MIN_STOCK_ALLOWANCE - sum;
            sum = 0.0;
            n = 0;
            m++;
          }
}

/* ************************************************************************
**
** FUNCTION solving_primary_tol
** Assign the primary operation tolerances using linear programming.
**
*/
void solving_primary_tol(void)
{
FILE *in_file;
int i, j, k, k1, kk, icase, *izrov, *iposv, m, n, m1, m2, m3, nm1m2;
int ftemp[MAX_NODE+1];
float s[ROW][COLUMN];
float **a;

          for(k1=0; k1<=NODE_NUM; k1++)
              ftemp[k1]=F_O_D[k1];

          in_file = fopen("tolp.dat", "r");
          if(in_file == NULL) {
            printf("\n The file can not be opened.");
          }
          else {
            fscanf(in_file, "%d%d%d%d%d\n", &m,&n,&m1,&m2,&m3);
            for(i=0; i<=m; i++)
              for(j=0; j<=n; j++)
```

```
            fscanf(in_file, "%f", &s[i][j]);
        }

        nm1m2 = n+m1+m2;
        izrov = ivector(1, n);
        iposv = ivector(1, m);
        a = convert_matrix(&s[0][0], 1, ROW, 1, COLUMN);
        simplx(a, m, n, m1, m2, m3, &icase, izrov, iposv);
        if(icase == 1) {
          printf("\n unbounded objective function \n");
          beep();
        }
        else if(icase == -1) {
          printf("\n no solution satisify constraints given\n");
          beep();
        }
        else  {
         k = 0;
         kk = 2;
         for(i=1; i<=m+1; i++)
           if(i == 1 || iposv[i-1] <= nm1m2)  {
                 if(i>=2 && iposv[i-1] <= n )  {
                  for(k1=0; k1<=NODE_NUM; k1++)  {
                   if(ftemp[k1] != 0 ) {
                     p[k1+iposv[i-1]-kk].tol = a[i][1];
                     ftemp[k1] = 0;
                     k1 = NODE_NUM;
                     kk++;
                   }
                  }
                  b[k] = a[i][1];
                  k++;
                 }
           }
    }
    fclose(in_file);
}

/* ************************************************************************
**
** FUNCTION solving_secondary_tol
** Assign the secondary operation tolerances using linear programming.
**
*/

void solving_secondary_tol(void)
{
FILE *in_file;
int i, ii, j, jj, k, k1, kk, icase, *izrov, *iposv, m, n, m1, m2, m3, nm1m2;
float ss[ROW][COLUMN];
int ftemp[MAX_NODE+1];
```

```
float **aa;

        for(k1=0; k1<=NODE_NUM; k1++)
           ftemp[k1]=F_O_ST[k1];


        in_file = fopen("tols.dat", "r");
        if(in_file == NULL)  {
          printf("\n The file can not be opened.");
        }
        else  {
          fscanf(in_file, "%d%d%d%d%d\n", &m,&n,&m1,&m2,&m3);
          for(i=0; i<=m; i++)
            for(j=0; j<=n; j++)
              fscanf(in_file, "%f", &ss[i][j]);
        }

        /* check the datafile */
        for(i=1; i<=m; i++) {
          k=0;
          for(j=1; j<=n; j++)  {
           if(ss[i][j] == 0.0 )
              k++;
          }
          if(k==n) {
            for(ii=i; ii<=m; ii++)
              for(jj=0; jj<=n; jj++)
                    ss[ii][jj]=ss[ii+1][jj];
          m--;
          m1--;
          i--;
          }
        }

        nm1m2 = n+m1+m2;
        izrov = ivector(1, n);
        iposv = ivector(1, m);
        aa = convert_matrix(&ss[0][0], 1, ROW, 1, COLUMN);
        simplx(aa, m, n, m1, m2, m3, &icase, izrov, iposv);
        if(icase == 1)
          printf("\n unbounded objective function \n");
        else if(icase == -1)
          printf("\n no solution satisify constraints given\n");
        else  {
          k = 0;
          kk = 2;
          for(i=1; i<=m+1; i++)
            if(i == 1 || iposv[i-1] <= nm1m2)  {
                  if(i>=2 && iposv[i-1] <= n)  {
          for(k1=0; k1<=NODE_NUM; k1++)  {
                      if(ftemp[k1] != 0 && F_O_D[k1]==0) {
```

```
                        p[k1+iposv[i-1]-kk].tol = aa[i][1];
                        ftemp[k1] = 0;
                        k1 = NODE_NUM;
                        kk++;
                    }
                    }
                    b[k] = aa[i][1];
                    k++;
                }
            }
        }
    fclose(in_file);
}

/* ************************************************************************
**
** FUNCTION simplx
** Simplex method for linear programming.
**
*/
void simplx(float **a, int m, int n, int m1, int m2, int m3,
                int *icase, int izrov[], int iposv[])
{
int i, ip, ir, is, k, kh, kp, m12, nl1, nl2;
int *l1, *l2, *l3;
float q1, bmax;


        if(m != (m1 + m2 + m3))
          printf("Bad input constraint counts in simplx");
        l1 = ivector(1, n+1);
        l2 = ivector(1, m);
        l3 = ivector(1, m);
        nl1 = n;
        for(k=1; k<=n; k++)
          l1[k] = izrov[k] = k;
        nl2 = m;
        for(i=1; i<=m; i++) {
          if(a[i+1][1] < 0.0)
            printf("Bad input tableau in simplx");
          l2[i] = i;
          iposv[i] = n + i;
        }
        for(i=1; i<=m2; i++)
          l3[i] = 1;
        ir = 0;
        if(m2+m3) {
          ir = 1;
          for(k=1; k<=n+1; k++) {
            q1 = 0.0;
            for(i=m1+1; i<=m; i++)
```

```
            q1 +=a[i+1][k];
        a[m+2][k] = -q1;
    }
    do {
        simp1(a, m+1, l1, nl1, 0, &kp, &bmax);
        if(bmax <= EPS && a[m+2][1] < -EPS) {
            *icase = -1;
            return;
        }
        else if(bmax <= EPS && a[m+2][1] <= EPS) {
            m12 = m1 + m2 + 1;
            if(m12 <= m) {
              for(ip=m12; ip <= m; ip++) {
                if(iposv[ip] == (ip+n)) {
                    simp1(a, ip, l1, nl1, 1, &kp, &bmax);
                    if(bmax > 0.0)
                      goto one;
                  }
              }
            }
            ir=0;
            --m12;
            if(m1+1 <= m12)
              for(i=m1+1; i<=m12; i++)
                  if(l3[i-m1] == 1)
                    for(k=1; k<=n+1; k++)
                      a[i+1][k] = -a[i+1][k];
            break;
        }
        simp2(a, n, l2, nl2, &ip, kp, &q1);
        if( ip == 0) {
            *icase = -1;
            return;
        }
one:    simp3(a, m+1, n, ip, kp);

        if(iposv[ip] >= (n+m1+m2+1)) {
            for(k=1; k<=nl1; k++)
              if(l1[k] == kp) break;
            --nl1;
            for(is=k; is<=nl1; is++)
              l1[is] = l1[is+1];
            ++a[m+2][kp+1];
            for(i=1; i<=m+2; i++)
              a[i][kp+1] = -a[i][kp+1];
        }
        else {
            if(iposv[ip] >= (n+m1+1)) {
              kh = iposv[ip] - m1 -n;
              if(l3[kh]) {
                l3[kh] = 0;
```

```
                    ++a[m+2][kp+1];
                    for(i=1; i<=m+2; i++)
                        a[i][kp+1] = -a[i][kp+1];
                  }
                 }
              }
              is = izrov[kp];
              izrov[kp] = iposv[ip];
              iposv[ip] = is;
           } while(ir);
         }
    for(;;) {
          simp1(a, 0, ll, nll, 0, &kp, &bmax);
           if(bmax <= 0.0) {
             *icase = 0;
             return;
           }
           simp2(a, n, l2, nl2, &ip, kp, &q1);
           if(ip == 0) {
             *icase = 1;
             return;
           }
           simp3(a, m, n, ip, kp);
           is = izrov[kp];
           izrov[kp] = iposv[ip];
           iposv[ip]=is;
         }
}

/* ************************************************************************
**
** FUNCTION simp1
** Determine the maximum of those elements whose index is contented in the
** supplied list ll either with or without taking the absolute value.
**
*/
void simp1(float **a, int mm, int ll[], int nll, int iabf,
                    int *kp, float *bmax)
{
int k;
float test;

          *kp=ll[1];
          *bmax=a[mm+1][*kp+1];
          for(k=2; k<=nll; k++) {
            if(iabf == 0)
              test = a[mm+1][ll[k]+1] - (*bmax);
            else
              test = fabs(a[mm+1][ll[k]+1]) -fabs(*bmax);
            if(test > 0.0) {
              *bmax = a[mm+1][ll[k]+1];
```

```
            *kp = ll[k];
          }
        }
}

/* ************************************************************************
**
** FUNCTION simp2
** Locate a pivot element, taking degeneracy into account.
**
*/
void simp2(float **a, int n, int l2[], int nl2, int *ip, int kp, float *q1)
{
int k, ii, i;
float qp, q0, q;

        *ip = 0;
        for(i=1; i<=nl2; i++) {
          if(a[l2[i]+1][kp+1] < -EPS) {
            *q1 = -a[l2[i]+1][1]/a[l2[i]+1][kp+1];
            *ip=l2[i];
            for(i=i+1; i<=nl2; i++) {
              ii=l2[i];
              if(a[ii+1][kp+1] < -EPS) {
                    q = -a[ii+1][1]/a[ii+1][kp+1];
                    if(q<*q1) {
                      *ip=ii;
                      *q1=q;
                    }
                    else if(q == *q1) {
                      for(k=1; k<=n; k++) {
                            qp= -a[*ip+1][k+1]/a[*ip+1][kp+1];
                            q0= -a[ii+1][k+1]/a[ii+1][kp+1];
                            if(q0 != qp)
                              break;
                      }
                      if(q0 < qp)
                        *ip = ii;
                    }
            }
          }
        }
    }
}

/* ************************************************************************
**
** FUNCTION simp3
** Matrix operations to exchange a left-hand and right-hand variable.
**
*/
```

```
void simp3(float **a, int i1, int k1, int ip, int kp)
{
int kk, ii, i, j;
float piv;

        piv = 1.0/a[ip+1][kp+1];
        for(ii=1; ii<=i1+1; ii++)
          if(ii-1 != ip)  {
            a[ii][kp+1] *= piv;
            for(kk=1; kk<=k1+1; kk++)
                  if(kk-1 != kp)
                    a[ii][kk] -= a[ip+1][kk]*a[ii][kp+1];
          }
          for(kk=1; kk<=k1+1; kk++)
            if(kk-1 != kp)
                  a[ip+1][kk] *= -piv;
            a[ip+1][kp+1]=piv;

}

/* ************************************************************************
**
** FUNCTION **ivector
** Allocate an int vector with range [nl...nh].
**
*/
int *ivector(long nl, long nh)
{
int *v;

        v = (int *)malloc((size_t)((nh-nl+2)*sizeof(int)));
        if(!v) {
          printf("\n allocation failurre in ivector()");
          exit(1);
        }
        return v-nl+1;
}

/* ************************************************************************
**
** FUNCTION **convert_matrix
** Allocate a float matrix m[nrl...nrh][ncl...nch]
**
*/
float **convert_matrix(float *a, long nrl, long nrh, long ncl, long nch)
{
        long i, j, nrow=nrh-nrl+1, ncol=nch-ncl+1;
        float **m;

        m=(float **)malloc((size_t)((nrow+1)*sizeof(float*)));
        if(!m) {
```

```
            printf("\n allocation failure in convert_matrix()");
            exit(0);
          }
          m +=1;
          m -=nrl;

          m[nrl]=a-ncl;
          for(i=1, j=nrl+1; i<nrow; i++, j++)
            m[j] = m[j-1] + ncol;
          return m;
}

/* ************************************************************************
**
** FUNCTION BEEP
** Make a short beep when called.
**
*/
void beep(void)
{
 sound(50);
 delay(200);
 nosound();
}

/* ************************************************************************
**
**  FUNCTION stock_removal_tol
**  Calculate the tolerance of each stock removal
**
*/
void stock_removal_tol(void)
{
int i, j, k;

          k = 0;
          for(i=0; i<NODE_NUM; i++)  {
            p[i].stock_tol = 0.0;
            if(p[i].stock_node != 0 )  {
              for(j=0; j<10; j++)  {
                    if(st_chain[k][j] != 0)
                      p[i].stock_tol += p[st_chain[k][j]-1].tol;

              }
              printf("\n%10.5f", p[i].stock_tol);
              k++;
            }
          }
}

/* ************************************************************************
```

```
**
** FUNCTION coefficient_matrix
** Determine the sign of operations in the tolerance chains and the
** coefficient matrix of the working dimension equation set.
*/
void coefficient_matrix(void)
{
int i, j, m, n;
FILE *out_file, *fptr;

        for(i=0; i<NODE_NUM-1; i++)
          for(j=0; j<NODE_NUM-1; j++)
            c[i][j] = 0.0;

        j = 0;
        for(i=0; i<NUMBER_OF_ST; i++)  {
          if(p[st_chain[i][j]-1].s_direction == -1) {     /* left side */
            while(st_chain[i][j] != 0) {
             if(j == 0) {
              if(p[st_chain[i][j]-1].s_order >
                          p[p[st_chain[i][j]-1].pre_node-1].s_order)
                   c[i][p[st_chain[i][j]-1].node-2] = 1.0;
              else
                   c[i][p[st_chain[i][j]-1].node-2] = -1.0;
              j++;
             }
             if(j >= 1 && p[st_chain[i][j-1]-1].pre_node ==
                                          p[st_chain[i][j]-1].node) {
              if(p[st_chain[i][j]-1].s_order >
                          p[p[st_chain[i][j]-1].pre_node-1].s_order)
                   c[i][p[st_chain[i][j]-1].node-2] = 1.0;
              else
                   c[i][p[st_chain[i][j]-1].node-2] = -1.0;
              j++;
             }
             if(j >= 1 && p[st_chain[i][j-1]-1].pre_node ==
                                          p[st_chain[i][j]-1].pre_node) {
              if(p[p[st_chain[i][j]-1].pre_node-1].s_order >
                                          p[st_chain[i][j]-1].s_order)
                   c[i][p[st_chain[i][j]-1].node-2] = 1.0;
              else
                   c[i][p[st_chain[i][j]-1].node-2] = -1.0;
              j++;
             }
          if(j>=1 && st_chain[i][j] != 0 &&
                          p[st_chain[i][j]-1].pre_node == st_chain[i][j-1]) {
               if(p[st_chain[i][j]-1].s_order >
                                p[p[st_chain[i][j]-1].pre_node-1].s_order)
                c[i][p[st_chain[i][j]-1].node-2] = -1.0;
               else
                c[i][p[st_chain[i][j]-1].node-2] = 1.0;
```

```
      j++;
     }
    }
    j = 0;
   }

   if(p[st_chain[i][j]-1].s_direction == 1) {   /* right side */
    while(st_chain[i][j] != 0) {
     if(j == 0) {
      if(p[st_chain[i][j]-1].s_order >
                    p[p[st_chain[i][j]-1].pre_node-1].s_order)
           c[i][p[st_chain[i][j]-1].node-2] = -1.0;
      else
           c[i][p[st_chain[i][j]-1].node-2] = 1.0;
      j++;
     }
     if(j >= 1 &&  p[st_chain[i][j-1]-1].pre_node ==
                                        p[st_chain[i][j]-1].node) {
      if(p[st_chain[i][j]-1].s_order >
                    p[p[st_chain[i][j]-1].pre_node-1].s_order)
           c[i][p[st_chain[i][j]-1].node-2] = -1.0;
      else
           c[i][p[st_chain[i][j]-1].node-2] = 1.0;
      j++;
     }
     if(j >= 1 && p[st_chain[i][j-1]-1].pre_node ==
                                p[st_chain[i][j]-1].pre_node) {
      if(p[p[st_chain[i][j]-1].pre_node-1].s_order >
                                        p[st_chain[i][j]-1].s_order)
           c[i][p[st_chain[i][j]-1].node-2] = -1.0;
      else
           c[i][p[st_chain[i][j]-1].node-2] = 1.0;
      j++;
     }
 if(j>=1 && st_chain[i][j] != 0 &&
                    p[st_chain[i][j]-1].pre_node == st_chain[i][j-1]) {
          if(p[st_chain[i][j]-1].s_order >
                                p[p[st_chain[i][j]-1].pre_node-1].s_order)
           c[i][p[st_chain[i][j]-1].node-2] = 1.0;
          else
           c[i][p[st_chain[i][j]-1].node-2] = -1.0;
      j++;
     }
    }
    j = 0;
   }
  }

  j = 0;
  for(i=0; i<NUMBER_OF_D; i++)  {
   if(p[listcn[i][j]-1].d_direction == -1) {
```

```
    while(listcn[i][j] != 0) {
     if(j == 0) {
      if(p[listcn[i][j]-1].s_order >
                    p[p[listcn[i][j]-1].pre_node-1].s_order)
          c[i+NUMBER_OF_ST][p[listcn[i][j]-1].node-2] = -1.0;
      else
          c[i+NUMBER_OF_ST][p[listcn[i][j]-1].node-2] = 1.0;
      j++;
     }
     if(j >= 1 && p[listcn[i][j-1]-1].pre_node ==
                                p[listcn[i][j]-1].node) {
      if(p[listcn[i][j]-1].s_order >
                    p[p[listcn[i][j]-1].pre_node-1].s_order)
          c[i+NUMBER_OF_ST][p[listcn[i][j]-1].node-2] = -1.0;
      else
          c[i+NUMBER_OF_ST][p[listcn[i][j]-1].node-2] = 1.0;
      j++;
     }
     if(j >= 1 && p[listcn[i][j-1]-1].pre_node ==
                                p[listcn[i][j]-1].pre_node) {
      if(p[p[listcn[i][j]-1].pre_node-1].s_order >
                                    p[listcn[i][j]-1].s_order)
          c[i+NUMBER_OF_ST][p[listcn[i][j]-1].node-2] = -1.0;
      else
          c[i+NUMBER_OF_ST][p[listcn[i][j]-1].node-2] = 1.0;
      j++;
      }
if(j>=1 && listcn[i][j] != 0 &&
                    p[listcn[i][j]-1].pre_node == listcn[i][j-1]) {
         if(p[listcn[i][j]-1].s_order >
                                p[p[listcn[i][j]-1].pre_node-1].s_order)
          c[i+NUMBER_OF_ST][p[listcn[i][j]-1].node-2] = 1.0;
         else
          c[i+NUMBER_OF_ST][p[listcn[i][j]-1].node-2] = -1.0;
      j++;
      }
    }
    j = 0;
   }

   if(p[listcn[i][j]-1].d_direction == 1) {
    while(listcn[i][j] != 0) {
     if(j == 0) {
      if(p[listcn[i][j]-1].s_order >
                    p[p[listcn[i][j]-1].pre_node-1].s_order)
          c[i+NUMBER_OF_ST][p[listcn[i][j]-1].node-2] = 1.0;
      else
          c[i+NUMBER_OF_ST][p[listcn[i][j]-1].node-2] = -1.0;
      j++;
     }
     if( j >= 1 &&
```

```
        p[listcn[i][j-1]-1].pre_node == p[listcn[i][j]-1].node) {
        if(p[listcn[i][j]-1].s_order >
                p[p[listcn[i][j]-1].pre_node-1].s_order)
         c[i+NUMBER_OF_ST][p[listcn[i][j]-1].node-2] = 1.0;
     else
        c[i+NUMBER_OF_ST][p[listcn[i][j]-1].node-2] = -1.0;
     j++;
    }
    if( j >= 1 && p[listcn[i][j-1]-1].pre_node ==
                        p[listcn[i][j]-1].pre_node) {
     if(p[p[listcn[i][j]-1].pre_node-1].s_order >
                                p[listcn[i][j]-1].s_order)
        c[i+NUMBER_OF_ST][p[listcn[i][j]-1].node-2] = 1.0;
     else
        c[i+NUMBER_OF_ST][p[listcn[i][j]-1].node-2] = -1.0;
     j++;
     }
     if(j>=1 && listcn[i][j] != 0 &&
                p[listcn[i][j]-1].pre_node == listcn[i][j-1]) {
        if(p[listcn[i][j]-1].s_order >
                        p[p[listcn[i][j]-1].pre_node-1].s_order)
         c[i+NUMBER_OF_ST][p[listcn[i][j]-1].node-2] = -1.0;
        else
         c[i+NUMBER_OF_ST][p[listcn[i][j]-1].node-2] =  1.0;
     j++;
     }
   }
   j = 0;
  }
}

out_file = fopen("c.dat", "w");
if( out_file == NULL) {
  printf("\n coeff. datafile can't open.");
  exit(3);
}
else {
  for(i=0; i<NODE_NUM-1; i++) {
    fprintf(out_file, "\n");
     for(j=0; j<NODE_NUM-1; j++)
        fprintf(out_file, "%5.2f ", c[i][j] );
  }
}


i=0;
j=0;
while(i<NUMBER_OF_ST) {
   if(p[j].stock_node != 0 ) {
     d[i] = p[j].stock_removal;
     i++;
```

```
            j++;
           }
           else
            j++;
        }

        i=0;
        j=0;
        while(i<NUMBER_OF_D) {
           if(p[j].dim != 0.0 ) {
            d[i+NUMBER_OF_ST] = p[j].dim;
            i++;
            j++;
           }
           else
            j++;
        }

        fptr = fopen("b.dat", "w");
        if( fptr == NULL) {
          printf("\n right-hand-side datafile can't open.");
          exit(3);
        }
        else {
          for(i=0; i<NODE_NUM-1; i++)
           fprintf(fptr, "%8.5f ", d[i]);
        }
}

/* ************************************************************************
**
** FUNCTION solving_equation
** Determine the working dimensions using Gaussian elimination.
**
*/
void solving_equation(void)
{
float scale, bt, e, x[MAX_NODE-1];
int i,j, l, k, flag;

        flag = 0;
        for(k=0; k<NODE_NUM-2; k++) {
          if(c[k][k] != 0.0) {
            for(i=k+1; i<NODE_NUM-1; i++) {
                  scale = c[i][k]/c[k][k];
                  for(j=k+1; j<NODE_NUM-1; j++)
                    c[i][j] = c[i][j] - scale*c[k][j];
                  d[i]=d[i]-scale*d[k];
            }
          }
          if(c[k][k] == 0.0) {
```

```
            for(l=k+1; l<NODE_NUM-1; l++) {
                  if(c[l][k] != 0.0) {
                   flag = 1;
                   for(j=k; j<NODE_NUM-1; j++) {
                    bt=c[k][j];
                    c[k][j]=c[l][j];
                    c[l][j]=bt;
                   }
                    bt=d[k];
                    d[k]=d[l];
                    d[l]=bt;

                   for(i=k+1; i<NODE_NUM-1; i++) {
                    scale = c[i][k]/c[k][k];
                    for(j=k+1; j<NODE_NUM-1; j++)
                          c[i][j] = c[i][j]-scale*c[k][j];
                    d[i]= d[i]-scale*d[k];
                   }
                   if(flag == 1)
                    break;
                   else {
                      printf("\n No unique solution.");
                      exit(2);
                   }
                  }
              }
            }
          }

          x[NODE_NUM-2]= d[NODE_NUM-2]/c[NODE_NUM-2][NODE_NUM-2];
          for(k=NODE_NUM-3; k>=0; k--) {
            e=0.0;
            for(j=k+1; j<NODE_NUM-1; j++) {
              e +=c[k][j]*x[j];
              x[k]=(d[k]-e)/c[k][k];
            }
          }

          for(i=0; i<NODE_NUM-1; i++) {
            p[i+1].wdim = x[i];
          }
}

/* ************************************************************************
**
** FUNCTION save_output
** Save the tolerance chart in the file "tolchart.dat".
**
*/
void save_output(void)
{
```

```
FILE *out_file;
int i;
int j, temp, tp[MAX_NODE], k[MAX_NODE];


        out_file = fopen("tchart.dat", "w");
        if( out_file == NULL) {
          printf("\n tolchart.dat file can't open.");
          exit(3);
        }
        else {
          fprintf(out_file, "%25s %16s\n",
                            " ", "TOLERANCE CHART");
          fprintf(out_file, "%25s %16s\n",
                            " ","====================");
          fprintf(out_file, "%5s %10s %10s %10s %10s %10s\n",
          " ","OPRT NO", "WORK DIM", "TOLERANC", "STOCK REM", "TOLERANCE");
          fprintf(out_file,"%5s %50s\n"," ",
          "-----------------------------------------------------------");
        }

        for(i=0; i<NODE_NUM; i++)
          tp[i]=p[i].oprt;

        j = 0;
        while(j<NODE_NUM) {
          temp = 999;
          for(i=0; i<NODE_NUM; i++) {
            if(temp > tp[i]) {
              temp = tp[i];
              k[j] = i;
            }
          }
          tp[k[j]] = 1000;
          j++;
        }

        for(i=1; i<NODE_NUM; i++) {
          if(p[k[i]].stock_removal == 0.0)
            fprintf(out_file, "%5s %10d %10.4f %10.4f %10s %10.4f\n", " ",
                              p[k[i]].oprt, p[k[i]].wdim, p[k[i]].tol,
                              "SOLID", p[k[i]].stock_tol);
          else
            fprintf(out_file, "%5s %10d %10.4f %10.4f %10.4f %10.4f\n", " ",
                              p[k[i]].oprt, p[k[i]].wdim, p[k[i]].tol,
                              p[k[i]].stock_removal, p[k[i]].stock_tol);
        }
        fprintf(out_file,"%5s %50s\n"," ",
          "-----------------------------------------------------------");
        fprintf(out_file,"%5s %16s %20s%16s\n"," ","BLUEPRINT","", "RESULTANT");
        fprintf(out_file, "%5s%50s\n", " ",
```

```
        "=================================================================");
        fprintf(out_file,"%5s %10s %10s %16s %10s %10s\n", " ","DIMENSION",
                "TOLERANCE", "", "DIMENSION", "TOLERANCE");
        fprintf(out_file, "%5s %50s\n"," ",
          "-------------------------------------------------------");
        for(i=0; i<NODE_NUM; i++) {
          if(p[i].dim_node != 0)
            fprintf(out_file, "%4s %10.4f%10.4f%19s%10.4f%10.4f\n",
            " ", p[i].dim, p[i].dtol, "", p[i].dim, p[i].process_tol);
          else
            i++;
        }
        fprintf(out_file, "%5s %50s\n"," ",
          "-------------------------------------------------------");

        fclose(out_file);
}

/* ************************************************************************
**
** FUNCTION display_output
** Display the tolerance chart on the screen.
**
*/
void display_output(void)
{
int i;
int j, temp, tp[MAX_NODE], k[MAX_NODE];

          clrscr();
          printf("%25s %16s\n"," ", "TOLERANCE CHART");
          printf("%25s %16s\n", " ","==================");
          printf("%5s %10s %10s %10s %10s %10s\n",
          " ","OPRT NO", "WORK DIM", "TOLERANC", "STOCK REM", "TOLERANCE");
          printf("%5s %50s\n"," ",
          "-------------------------------------------------------");

        for(i=0; i<NODE_NUM; i++)
          tp[i]=p[i].oprt;

        j = 0;
        while(j<NODE_NUM) {
          temp = 999;
          for(i=0; i<NODE_NUM; i++) {
            if(temp > tp[i]) {
              temp = tp[i];
              k[j] = i;
            }
          }
          tp[k[j]] = 1000;
          j++;
```

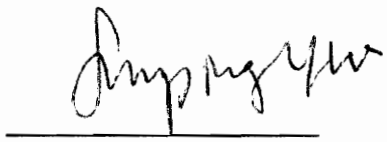
```
        }

        for(i=1; i<NODE_NUM; i++) {
          if(p[k[i]].stock_removal == 0.0)
            printf("%5s %10d %10.4f %10.4f %10s %10.4f\n", " ",
                        p[k[i]].oprt, p[k[i]].wdim, p[k[i]].tol,
                        "SOLID", p[k[i]].stock_tol);
          else
            printf("%5s %10d %10.4f %10.4f %10.4f %10.4f\n", " ",
                        p[k[i]].oprt, p[k[i]].wdim, p[k[i]].tol,
                        p[k[i]].stock_removal, p[k[i]].stock_tol);
        }
        printf("%5s %50s\n"," ",
          "-------------------------------------------------------------");
        printf("%5s %16s %20s%16s\n"," ","BLUEPRINT","", "RESULTANT");
        printf("%5s%50s\n", " ",
        "=============================================================================");
        printf("%5s %10s %10s %16s %10s %10s\n", " ","DIMENSION",
                "TOLERANCE", "", "DIMENSION", "TOLERANCE");
        printf("%5s %50s\n"," ",
          "-------------------------------------------------------------");
        for(i=0; i<NODE_NUM; i++) {
          if(p[i].dim_node != 0)
            printf("%4s %10.4f%10.4f%19s%10.4f%10.4f\n",
            " ", p[i].dim, p[i].dtol, "", p[i].dim, p[i].process_tol);
          else
            i++;
        }
    printf("%5s %50s\n"," ",
          "-------------------------------------------------------------");
        printf("\n");
        printf("PRESS ANY KEY TO CONTINUE.");
        getch();
}

/* ************************************************************************
**
** FUNCTION instruction()
** Ask the user if they would like instructions, if so display them.
**
*/
void instruction(void)
{
 int key;

   clrscr();
   setcbrk(1);
   printf("\n\nWould you like instructions Y/[N] ?");
   key=getch();

   if((key==89)||(key==121))
```

```
    {
      clrscr();
      printf("Loading instructions . . .\n\n");
      system("TYPE INSTRC.DOC|MORE");
    }
    printf("\nLoading data . . .\n\n");
}

/*****************************************************************************
**
** Function calculating_design_tol
** Calculate the tolerance of each blueprint dimension
*/
void calculating_design_tol(void)
{
int i, j;
float tempt[MAX_DIM];

          for(i=0; i<MAX_DIM; i++)
            tempt[i] = 0.0;

          i=0;
          while(i<MAX_DIM) {
           for(j=0; j<M_NUM_OF_CHAIN; j++) {
             if(listcn[i][j] != 0)
               tempt[i] += p[listcn[i][j]-1].tol;
           }
           i++;
          }

          j=0;
          i=0;
          while(j<NODE_NUM) {
            if(p[j].dim_node != 0) {
              p[j].process_tol=tempt[i];
              j++;
              i++;
            }
            else
              j++;
          }
}□
```

VITA

Junping Yue was born on March 4, 1961, in Jili Province, China. He received his B.S. in Mechanical Engineering at Central South Forestry University in July 1982. He then worked for the Central South Design Institute for one year. In August 1983, he was admitted as a graduate student of Mechanical Engineering at Nanjing Forestry University and received his M.S. in August 1986. In the same year, he joined the faculty of Mechanical Engineering Department at Nanjing Forestry University as a Lecturer until he came to USA.

He first studied in Forest Engineering Department at University of Washington, Seattle, for five months. In August 1990 he enrolled in Forest Products Department at Virginia Polytechnic Institute and State University and was engaged in research related to production optimization and simulation. Because of his strong interest in manufacturing systems, he transferred into the Industrial and Systems Engineering Department in June 1993 to obtain his M.S. degree in Manufacturing Systems Engineering.



Junping Yue