INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI®

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Neural Network Based Shaped Neighborhoods: A Design Retrieval System

by

Frank S. Holman III

A dissertation submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

1999

Program Authorized to Offer Degree: Department of Electrical Engineering

In presenting this dissertation in partial fulfillment of the requirements for the Doctoral degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of the dissertation is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to University Microfilms, 300 North Zeeb Road, Ann Arbor, MI 48106-1346, to whom the author has granted "the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform."

Signature Frank S Holman III

Date 5/21/99

University of Washington
Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

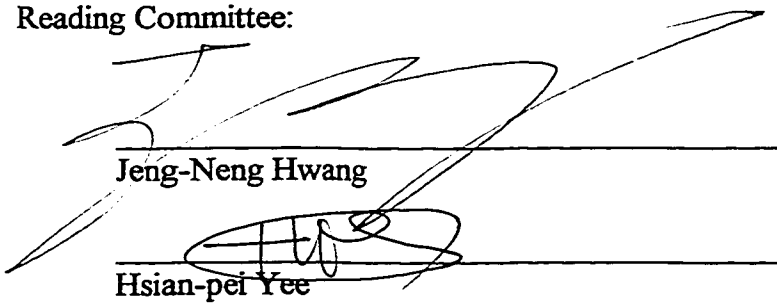
Frank S. Holman III

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Chair of Supervisory Committee:

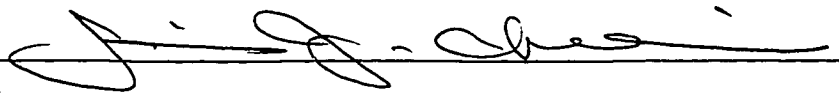
Robert J. Marks II

Reading Committee:



Jeng-Neng Hwang

Hsian-pei Yee



Jai Choi

Date: 5/21/99

University of Washington

Abstract

Neural Network Based Shaped Neighborhoods: A Design Retrieval System

by Frank S. Holman III

Chairperson of the Supervisory Committee:
Professor Robert J. Marks II
Department of Electrical Engineering

The life cycle cost of a single detail part in a large assembly such as an airframe includes design, tooling, manufacture, maintenance and other associated computing overhead throughout the 40 year life of the part. These expenses are a significant factor in the final cost of the assembled product. As such, opportunities exist in preliminary design to identify and exploit existing "similar parts" that are already configured in the release system. The notion of a similar part properties includes geometric parameters as well as other non-geometric attributes such as material and finish.

This dissertation is based on an 8-dimensional hyperspace where the axes of the space represent an appropriate set of the detail part properties discussed above. Thus, any given detail part can be mapped directly into this space. In this domain, the concept of similar part discovery reduces to identifying an appropriate neighborhood around a given target location. This dissertation develops "shaped neighborhoods" in hyperspace using a neural

network based vector approach. The method provides the required flexibility to define highly specific neighborhoods which in turn yield highly refined search results. Put another way, these 8-dimensional shaped neighborhoods, called "cookie cutters", provide the means for defining individual property variability as a simultaneous function of all other property variability.

Finally, an improved NN training algorithm specifically suited to the network topology used to represent the shaped neighborhoods is presented. In this algorithm, Kohonen self-organizing feature maps (SOFM) are used to define a topology preserving mapping between a high dimensional NN input space and a two dimensional output map space. NN training error is associated with respective SOFM map nodes. Thus, the Kohonen map nodes represent vector prototypes over the input domain and the values associated with each map node are representative of training error for a generalized region in the input domain. This dissertation exploits this relationship by defining a weight update expression based on this error leading to an accelerated network convergence algorithm.

TABLE OF CONTENTS

Page

List of Figures iv

List of Tables vii

Chapter 1: Introduction 1

1.1 Problem Statement 1

1.2 Previous Work 3

1.3 Contributions 3

1.4 Dissertation Outline 4

Chapter 2: Shaped Neighborhoods 6

2.1 Introduction 6

2.2 Motivation 7

2.3 Engineering Detail Part Types 8

2.4 Relational Database Infrastructure 11

2.5 Detail Part Parameterization in Hyperspace 12

2.6 Theoretical Specification of Shaped Neighborhoods 14

2.7 Design Retrieval Methodology 18

Chapter 3: Analytic Based Neighborhoods 24

3.1 Introduction 24

3.2 Relationships Through SQL Queries 24

3.3 Generalized Analytic Filters 26

Chapter 4: Fuzzy Logic Based Neighborhoods **29**

4.1 Introduction 29

4.2 Parametric Fuzzy Logic Comparison 30

4.3 Generalized Fuzzy Logic Based Shaped Neighborhoods 33

4.4 A Fuzzy Logic Shaped Neighborhood Case Example 36

Chapter 5: Neural Network Based Neighborhoods **39**

5.1 Introduction 39

5.2 Neural Network Topology 40

5.3 Network Training 42

5.4 Results 43

Chapter 6: Neighborhood Visualization Methods **46**

6.1 Introduction 46

6.2 Neural Network Based Ray Tracing 47

6.3 Code Analysis 49

6.4 TIFF Image of Shaped Neighborhood 52

Chapter 7: An Accelerated Convergence Algorithm **53**

7.1 Introduction 53

7.2 Neural Networks and Function Representation 54

7.3 Layered Perceptrons and Back Propagation 55

7.4 Convergence Criteria 57

7.5 Feature Map Based Training 57

7.6 Quick-Prop Comparison 63

7.7 Feature Map Visualization 64

7.8 Accelerated Convergence 65

7.9 Results 66

Chapter 8: A Detail Part Design Retrieval System 75

8.1 Introduction 75

8.2 System Architecture 76

8.3 The Physical Prototype 80

8.4 Results and verification 84

Chapter 9: Conclusion AND Future Work 88

9.1 Conclusion 88

9.2 Future Direction 88

Bibliography 90

Appendix A: Source Code for Shaped Neighborhood Visualization 93

Appendix B: C Code for Shaped Neighborhood Implementation 105

LIST OF FIGURES

Number page

2-1 Shear tie detail part 10

2-2 Part Properties 14

2-3 Union and Intersection 19

2-4 Neural Network based filters 20

2-5 Three-dimensional Neural Net 21

2-6 Stacked filters in hyperspace 22

3-1 Part properties 26

3-2 L-alpha norms 27

3-3 A family of neighborhoods 28

4-1 Overview of fuzzy logic filter 30

4-2 Fuzzy membership function 31

4-3 Antecedent/Consequence table 32

4-4 Defuzzification using central tendency 33

4-5 Fuzzy membership functions 34

4-6 Decision surface 35

4-7 Fuzzy Logic based shaped neighborhood 36

4-8. Two similar parts detected 37

4-9 A rejected part 38

5-1 Neural Network Topology 41

5-2 Six part selected 44

5-3 Three parts not selected 45

6-1 Modular, black box NN implementation 46

6-2 Neural network visualization 48

6-3 Network declarations 49

6-4 Network weights I/O code 50

6-5 Network weights available externally 50

6-6 Evaluate the NN at a specific point 51

6-7 Shaped Neighborhood showing artifact 52

7-1 NN architecture 56

7-2 Kohonen map unfolding 59

7-3 SOFM based image 60

7-4. SOFM based weight update 61

7-5 SOFM schematic 62

7-6 Quick-Prop parabola 63

7-7 Map progression 64

7-8 Graphical Visualization of Weight Evolution 65

7-9 Shaped neighborhood in 3D 66

7-10 NN convergence comparison 67

7-11 Q-Prop vs. SOFM convergence compare 70

8-1 Overview of Design Retrieval System 76

8-2 DB2 Relational Database table structure 77

8-3 Sample data used to load DB2 Database 80

8-4 SQL syntax 81

8-5 The C Code 83

8-6 Four selected parts 85

8-7 Two rejected parts 86

8-8 Target part and two selected parts 87

LIST OF TABLES

Number page

1. Detail Part Types 9
2. Hypercube Properties 13
3. Iris Classification Data 69

Chapter 1
Introduction

The life cycle cost of a single detail part in a large assembly (such as a commercial airframe) is accumulated from various processes including design, tooling, manufacture, maintenance and other associated computing overhead throughout the 40 year life of that part. These detail part expenses are a significant factor in the final cost of the assembled product. Opportunities exist in preliminary design to identify and exploit existing "similar detail parts" that are already configured in the release system, thereby decreasing the overall product cost by reducing or eliminating portions of the various processes listed above. The notion of a similar part in this context certainly includes geometric similarity, but also spans other non-geometric attributes such as material, finish, and other physical properties such as strength. The challenge is to organize the massive amounts of data associated with these detail parts, and to provide an efficient, intelligent design retrieval methodology, and ultimately avoid detail part redundancy.

1.1 Problem Statement

Current commercial airframe design and manufacture is typically organized in terms of functional divisions (structures, payloads, electrical, hydraulic, etc.), which typically have their own design tools and processes. Detail part designs and related supporting processes tend to get localized within these divisional boundaries. Even within a division, there are wide ranges of disciplines and computing environments impairing the sharing of design data. As such, a design engineer may design a bracket or clamp and release it to the

configuration management system unaware that an already released part (including all its process plans and manufacturing support data) was already available.

The problem is compounded by the very notion of part similarity. While it is true that certain design parameters such as strength and physical dimension are objective and dictated by the design requirements, it is also true that certain design attributes such as finish or the combination of material and thickness are subjective, and based on an engineer's training, experience, and preferences. The final design is a product of these objective and subjective parameters. Thus, the concept of similar part should include all the different geometric and non-geometric properties, plus the range of acceptable variability for each of these properties, plus a method for relating the range of variability between different properties of the design.

To pose the problem euphemistically, this dissertation is concerned with finding a needle (actually, a handful of needles) in a haystack. The needles are individual detail part designs and the haystack is a configuration management system (engineering release system) containing millions of detail parts. Note, the term detail part refers to an individual manufactured object which is typically associated with a part number. In classical configuration management terminology, detail parts are used on assemblies, and assemblies get incorporated into final products. The finding of the needles corresponds to identifying a reasonable collection of similar parts given some target design which could then be considered as candidates for reuse. The purposes of course is to eliminating the potential creation of a new detail part when a suitable similar part already exists. Thus a handful of similar part candidates might be the best 5 or 10 matches as determined by some criteria, which a design engineer could evaluate individually for suitability with respect to a particular design requirement.

1.2 Previous Work

With such obvious pay back associated with the reduction of duplicate parts in the aerospace industry and others, it is easy to understand the motivation for defining and implementing effective design retrieval systems. Certainly, the integration of relational databases with object oriented technology can provide an environment capable of addressing the issues associated with similar part identification and retrieval through class definitions supporting methods for similarity processing. Unfortunately, these systems are not yet up to the task of supporting millions of parts.

Neural Networks have been employed at Boeing Commercial Airplane company utilizing ART (Adaptive Resonance Theory) technology developed by Carpenter and Grossberg [1991]. These methods include pre-clustering detail parts with respect to their 2D profiles, and other physical properties. A user interface delivers these cluster in response to a user provided target part. The downside of this method is that similar part candidate must be pre-clustered in a relatively computing intensive training cycle which tends to encumber modern day concurrent engineering design methods.

1.3 Contributions

This dissertation defines and prototypes a highly responsive engineering design retrieval system suitable for implementation on a large scale supporting millions of part instances. In particular, innovations developed include the notion of shaped neighborhoods in high dimensional space as a vehicle for discriminating part instances with respect to generalized and functionally related part characteristics.

A number of software tools and methods were developed specifically to support the implementation of the design retrieval system defined in this dissertation. Specifically, a set of visualization tools capable of rendering shaped neighborhoods into TIFF images through a modified ray tracing algorithm is defined. While this tool is ideally suited for renderings of 3D shaped neighborhoods, it is general in nature and could ostensibly be used by any developer of neural network applications requiring visualization. Recall that high dimensional spaces have potentially useful projections into 3 dimensions.

Finally, a Kohonen [1984] Self-Organizing Feature Map (SOFM) is used to characterize neural network input space leading to a fuzzy logic based method for accelerated neural network convergence. This algorithm is developed especially to assist in training neural network topologies optimized for use as shaped neighborhoods in the context of the design retrieval system presented in this dissertation. And, like the visualization tools above, this technique is applicable to neural network training in general, providing convergence visualization in the open loop mode, and convergence acceleration in the closed loop, fuzzy logic feedback mode.

1.4 Dissertation Outline

Chapter 2 develops the concept of shaped neighborhoods in N-dimensional space specifically as applies to the design retrieval system defined in this dissertation. These neighborhoods are further put in context by establishing engineering detail part parameterizations, and the relevance of neighborhood concepts. Chapter 3 through 5 continue the motivation and development of shaped neighborhoods through analytic, fuzzy logic, and neural network based approaches. Chapter 6 presents a set of visualization tools capable of operating in the realm, thus providing enhanced

development capabilities as well as visual verification of results. An accelerated convergence algorithm including fuzzy logic feedback derived from Kohonen SOFM is presented in chapter 7. Finally, a description of the definition, implementation, and verification of an engineering detail part design retrieval prototype based on the methods and techniques developed in this dissertation is presented in chapter 8, followed by conclusions and future work in chapter 9.

Chapter 2
Shaped Neighborhoods

2.1 Introduction

This dissertation defines an Engineering Design Retrieval System based on Shaped Neighborhoods in high dimensional space. Detail Parts are characterized using their physical properties and mapped into the high dimensional space accordingly. Parts similar to a given target part are "discovered" by considering various neighborhoods surrounding the target part's represented in the space. A number of methodologies are suitable for representing these N-dimensional neighborhoods including analytic, fuzzy logic, and neural network approaches, where the "shape" of the neighborhood produced by these representations ultimately determines the set of similar parts returned by the method. Thus a given shaped neighborhood selects a region of space by virtue of its boundary definition, and points inside the boundary correspond to instances of detail parts considered similar to a target part located at the origin of the shaped neighborhood.

This chapter describes a computing environment based on an 8-dimension space for characterizing and retrieving individual engineering detail part designs. A motivation for such an approach is given followed by a definition of engineering detail parts as applies to this dissertation. Next, a relational database supporting infrastructure is motivated and developed. The hyperspace concept is introduced and a detail part parameterization in this space is defined A theoretical specification of a shaped neighborhood with respect to the computing environment is given, and finally a design retrieval methodology based on these shaped neighborhoods is presented.

2.2 Motivation

A number of factors motivate the shaped neighborhood based design retrieval system presented in this dissertation. Certainly, the shear magnitude (scale) of the problem domain is important in light of the fact that commercial airframes can contain millions of detail parts. Some object-relational systems may perform reasonably with total database entity counts in the tens of thousands but fail at the scale required by the commercial aircraft industry. A second factor is characterized as the ability to completely specify selectivity. Other retrieval systems can handle large quantities of data but sacrifice the ability to constrain selection criteria in a meaningful way. This shortcoming manifests itself when hundreds or thousands of parts are returned in response to some search criteria making it humanly impractical to inspect the results for final acceptance.

Large manufactured structures such as the Boeing 777 airframe, can incorporate millions of detail parts. These structures are typically designed in a hierarchical fashion along divisional (functional) disciplines where detail parts get used on assemblies which in turn get incorporated into larger and larger substructures. Divisions (structures, payloads, electrical, hydraulic, etc.) typically have their own design tools and processes, and detail part designs tend to get localized within these divisional boundaries. Studies have shown that part counts might be reduced up to 15% if a "discovery and part sharing mechanism" were in place during preliminary design. These studies are motivated by the large costs associated with designing, producing, and maintaining part configuration during the lifetime of the product. In the aerospace industry, this lifetime is typically mandated as 40 years or more by the Federal Aviation Agency (F.A.A).

The aerospace industry is in a constant state of change with respect to both airplane products and the computing environments which support the design and manufacture of those products. Dynamic advances in materials and engine technology, coupled with the ever present customer demand for new airplane products deemed necessary to support

various market strategies, results in a continuing procession of new airplane products and derivatives. Manufactures place incredible emphasis on time-to-market and other cost reduction strategies in order to compete for these market opportunities. One of the most important cost reduction strategies that can be employed is design retrieval, also known as part reuse, where suitable previously designed and released parts are identified and reused in two or more places. In this manner, the costs associated with design, tooling, manufacture, and maintenance are distributed over the number of individual uses for each part.

This dissertation develops a parametric approach to detail part representation and subsequent search, through the use of an N-dimensional hyperspace as defined in terms of classical linear spaces and their associated algebra. As such, a linear space is defined set theoretically in terms of its operations including addition and scalar multiplication where each axis of the space is associated with a specific detail part property (length, width, principal moments of inertia, etc.). Thus, detail parts are represented as points in this space, and similar parts tend to be neighbors in the space. Search mechanism become synonymous with defining and operating on neighborhood boundaries. Author B. B. Chaudhuri [1996] suggests a new definition of a point in multi-dimensional space, and their associated neighborhoods. He proposes choosing neighbors by minimizing the distances to the centroid of an evolving point aggregate thus creating a symmetry constraint. This definition is not appropriate to this dissertation due to this symmetry.

2.3 Engineering Detail Part Types

State-of-the-art airframe construction includes a hierarchical structure where major components such as wings, fuselage, and tail are built up from substructures such as frames, stringers, and ribs, which are in turn built up from detail parts. Some common

detail part types used in the manufacture of airframes are shown in Table 1. To complicate matters, many of these part types are produced automatically through Intelligent CAD (ICAD) systems using generative, parametric algorithms. As such, significant numbers of parts exist with only minor variations in certain non-critical properties such as fastener location or edge margins, etc. These parts are excellent candidates for a similar part discovery and reuse methodology.

Table 1. Detail Part Types

Angles	Fittings	Shear Ties
Stiffeners	Brackets	Floor Panels
Shims	Tees	Channels
Plates	Spacers	Clips
Tube Blocks	Duct Supports	Zees

There are actually hundreds of part types in a commercial airframe and each part type can have tens of thousands of families. Each family can have hundreds of members. Associated with each and every detail part is an extensive set of design, manufacture, and maintenance processes and their associated costs. The end result is that there are literally millions of detail parts being used on large airframes with an estimated reuse opportunity on up to 15% of these parts.

An example of a detail part from the shear tie part type is shown in Figure 2-1. As the name implies, this particular part type connects two or more surface type structures together at some shear angle. The part shown might secure a wing rib to the wing skin. The scallops actually provide weight reduction by eliminating certain non-structural metal from the part. Fasteners holes would be drilled corresponding to the scallop locations.

While the weight saved for this particular part due to scallops is measured in fractions of a gram, the weight mounts up when multiplied by the hundreds of thousands of detail parts on an airframe.

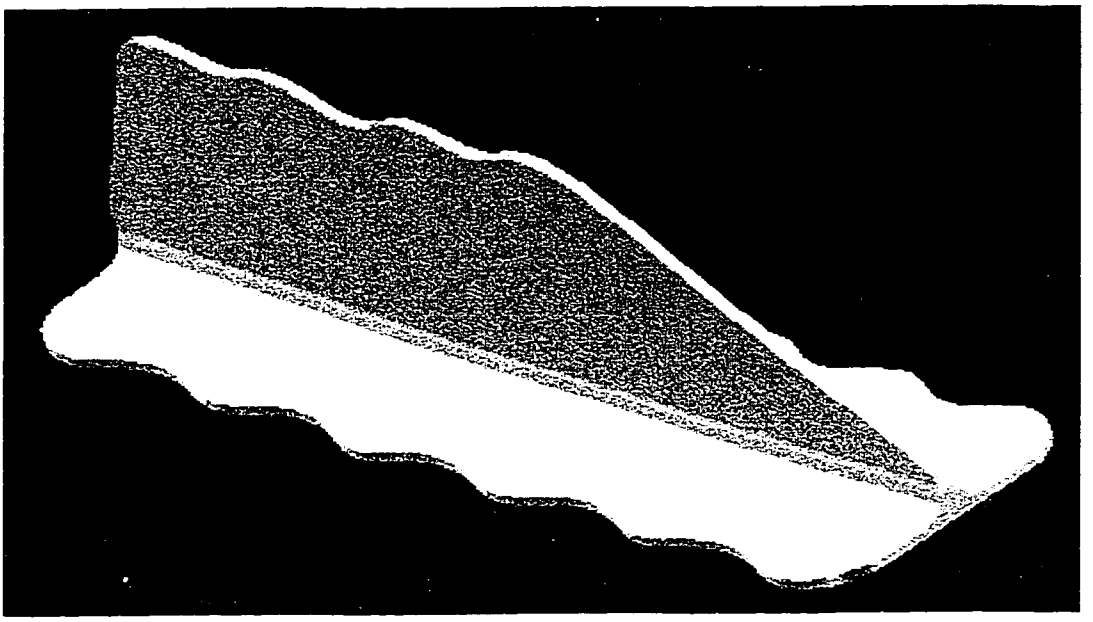


Figure 2-1. Shear tie detail part. Family members would vary parametrically in cross-section, length, material, etc.

Geometric parameters (length, thickness, etc.) and non-geometric attributes (material, finish, etc.) are defined for each detail part and stored in a computer based design environment. Taken together, they comprise the part's properties. The objective of a design retrieval system is to provide opportunities during preliminary design for engineers to discover and analyze the suitability of existing part design with respect to their current design requirements. This suitability is of course dependent on the part's properties, so the discovery mechanism must as a minimum take these into account. Further, a more refined discovery mechanism would also take into account relationships between properties such as strength-to-weight ratios etc.

2.4 Relational Database Infrastructure

Large scale Object-Relational databases have come of age. Invariably, these databases store data in tables which are related through keys. In particular, tables are typically 2 dimensional constructs where the columns of the table represent individual object parameters and the rows represent single instances of the object. Key attributes relate various tables together through a defined schema. Standard Query Language (SQL) is used almost exclusively as an access interface [Chamberlin, 1996]. SQL is an international standard supported by most database providers. It is an evolving language which supports a rich access to data.

The concept of fuzzy search is implemented in this language where a desired attribute value can be specified as a range. However, this is not really fuzzy in terms of a classical generalized membership function. Even though a given parameter is specified over a range, the boundaries of that range are crisp and the probability for a given parameter value over the range is uniform. As such, boundaries between selected and not selected instances in search space are linear. In fact, when two or more search parameters are specified as ranges, the resulting selection is rectilinear in search space. This may be acceptable in lower dimensions, but suffers considerably in higher dimensions with respect to the ability to produce refined discrimination over the domain.

The relational database, with its associated table structure and SQL capability, does however provide an excellent environment on which to implement the more sophisticated search methodologies presented as shaped neighborhoods in this dissertation. In particular, the IBM Universal Database DB2 for AIX workstations (known as UDB) provides terabytes of storage, and SQL query techniques are excellent for returning the rectilinear sub-regions in space corresponding to a first cut at part similarity.

2.5 Detail Part Parameterization in Hyperspace

The design retrieval system presented in this dissertation is founded on the idea that given an appropriate parameterization and representation of geometric detail part properties in some N-dimensional space, where N is the number of individual detail part parameters defined in the representation, similar parts will be "neighbors" in that space. This hyperspace is formed by a set of N orthogonal axes, where each axis is associated with a unique part parameter (e.g. length, mass, etc.), and where individual detail parts are represented as points in the hyperspace. The domain of the hypercube thus formed is simply the range along each axis determined by the minimum and maximum values for that particular part property exhibited by any member of the part population. Since the properties are all positive real values, the hypercube is entirely in the 1^{st} orthant. The individual detail part properties become a set of indexes which uniquely locate that part in the space.

Today's CAD systems provide a modeling environment featuring exact solid models. An entire airplane is digitally pre-assembled (defined) in this environment where form, fit, and function can be analyzed and tested without actually creating physical parts. Thus, detail parts are defined as "solids" in this environment, complete with physical attributes length and density. Computer methods for performing operating on these solids (translations, rotations, etc.) as well as ascertaining other physical properties such as volume, surface area, and moments of inertia are immediately available in the CAD environment. These tools are used to capture the required detail part properties used in this dissertation.

The hyperspace developed for this dissertation is 8 dimensional, each dimension representing a geometric property of detail parts. Parts are first placed in a "standard" orientation through use of their moments of inertia, where the major part axis is aligned

with the X axis of the defining coordinate system. Then the center of gravity is translated to the origin. A bounding box is defined based on this orientation and the height, length and width of this box become the first three indexes into the hypercube. The other properties shown complete the 8 dimensional coordinates of the point representation in hyperspace for any particular detail part. Table 2 shows these properties.

Table 2. Hypercube Properties

Dimension	Property
1	Bounding box height
2	Bounding box length
3	Bounding box width
4	Part surface area
5	Part volume
6	X moment of inertia (I_x)
7	Y moment of inertia (I_y)
8	Z moment of inertia (I_z)

Note that the long axis of the part aligns with I_x, the moment of inertia around the x axis. This is a result of the standard orientation routine applied to all database parts (and indeed all target parts) to facilitate comparisons. The eight real values obtained from this property analysis become the 8 indices into the hypercube and provide a unique "handle" on part geometry. In actual implementation, the values are stored in corresponding columns in the relational database part table. Similar parts are discovered by identifying a neighborhood around a target part's properties as representation in the hyperspace. These properties are shown more specifically for a shear tie detail part in Figure 2-2.

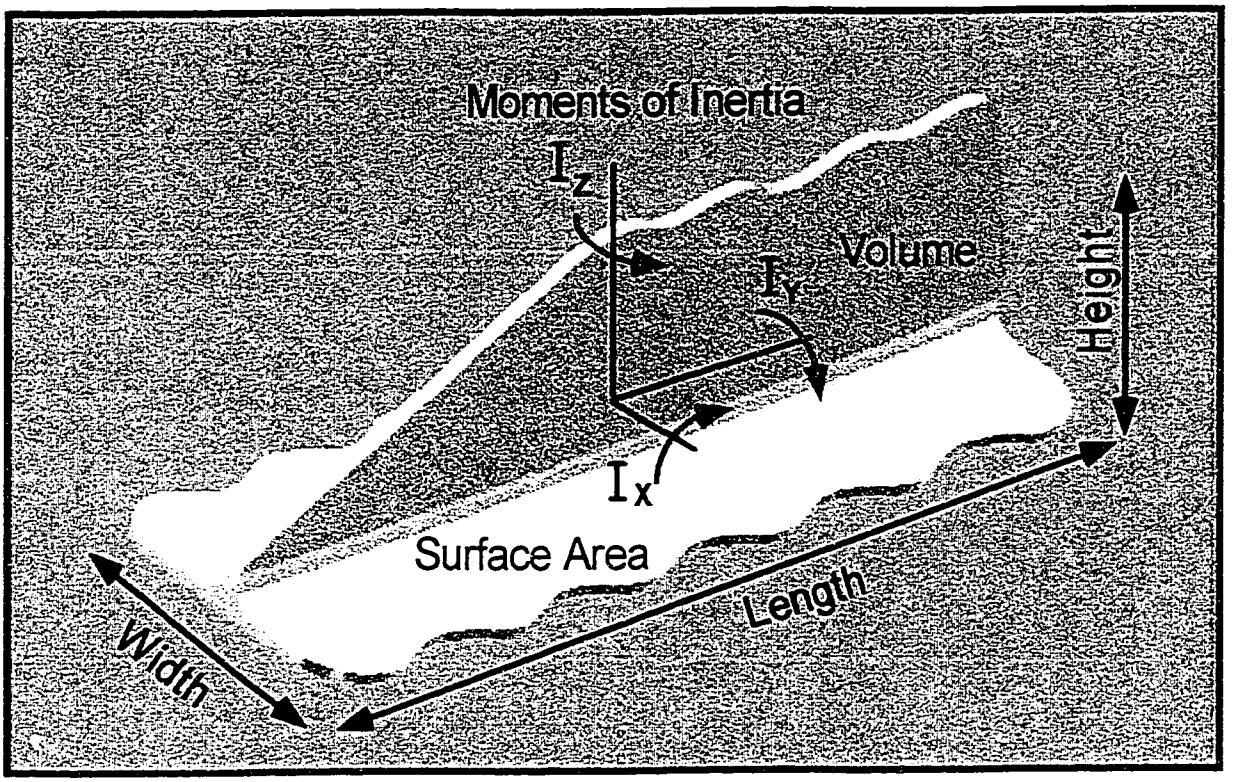


Figure 2-2. Part properties used as indexes into the hypercube.

2.6 Theoretical Specification of Shaped Neighborhoods

The notion of "similar part" now equates to the concept of a neighborhood around some target location in the hyperspace as defined above. This concept is extended to include the idea of "shaped neighborhoods" as opposed to simply a rectilinear or spherical neighborhood. This generalized shaped neighborhood is motivated by the desire to be able to define the neighborhood boundaries functionally, that is to say as functions between each parameter and all other parameters. This providing an environment able to fully exploit existing inter-parameter relationships, thereby extending the discriminating power of the system. These shaped neighborhoods are implemented using three interchangeable filter classes based on methodology, namely analytic, fuzzy logic, and neural networks.

All three classes of filters (described in this dissertation) can be considered black boxes at the functional level, that is, candidate parts are presented at the input and selected parts meeting the filter criteria are returned at the output.

At a theoretical level, an N-dimensional hyperspace is defined where each axis of the space is associated with a specific physical property of a detail part. Thus the domain for the space is limited by the maximum expected value for each of the properties. In practice, a bounding box is obtained for each detail part to be mapped into the hyperspace based on a standard orientation algorithm. Other property data is obtained either directly, or derived from, a computer representation of the part. Thus, a given detail part maps to a specific location in the space. The range on each of the principal axes in the hypercube is specific to the physical property being represented by that axis. For instance, the bounding box length axis would be real valued and go from zero to a couple hundred inches or so. In this manner, a database of parts is represented in hyperspace as discrete points whose coordinates in the space are given by the physical properties of each of the respective parts.

Finally, the candidate detail parts are discovered through a succession of filtering processes imposed on this hypercube structure. Classical SQL queries return a superset of parts which can be considered a first approximation of the desired similar part candidates. The rectilinear regions characteristic of such an SQL query typically deliver hundreds of candidate parts depending on the concentration in any particular region of the hypercube. This is usually more than a design engineer can handle during an evaluation phase. Consequently, the SQL superset is further refined through the use of analytic, fuzzy logic, or neural network based shaped neighborhood filters used to bring the final number of returned candidates down to a reasonable number.

There is an interesting feature of the hypercube regarding enclosed volumes. Imagine a sphere inside a bounding box in hyperspace. In 2 dimensions, this is just a circle in a square. The area outside the circle of radius r, yet inside the bounding square is given:

$$\textbf{Area} = (2r)^2 - \pi\,(r^2) \tag{1}$$

and the percent of area outside the circle yet inside the bounding square is given:

$$\textbf{\% Area} = ((2r)^2 - \pi\,(r^2)) \,/\, (2r)^2 \tag{2}$$

Now for the unit circle, r=1, the percentage is 21%. The equations for determining the volume for the "unit sphere" in a given n-dimensional space are given in (3) and (4).

$$V_{2n} = \frac{\pi^n r^{2n}}{n!} \tag{3}$$

$$V_{2n+1} = \frac{2^{2n+1}\, n!\, \pi^n r^{2n+1}}{(2n+1)!} \tag{4}$$

In 3 dimensions, the volume outside the unit sphere, yet inside the tangent bounding box is 47%. Extending the analogy to higher dimension yields surprising results which have a

profound effect on the type of searches considered in this dissertation. Namely, the volume for an 8 dimensional "sphere" of radius r is:

$$\textbf{Volume} = \pi^4 r^8 / 4! \qquad (5)$$

For an 8 dimensional ball of unit radius, this yields a volume of 4.06 and the percentage of "volume" outside the unit "sphere" yet inside the bounding "box" is given:

$$\textbf{\% Volume} = (512 - 4.06) / 512 = 99.2\% \qquad (6)$$

Equation (6) states that over 99% of all points in an 8 dimensional hypercube bounding a unit sphere in that space, are further away in magnitude than the unit norm from the center of the cube. Another way of saying this is that over 99% of the bounding box volume is outside the unit sphere centered in that box. Often referred to as "The Curse of Dimensionality", this has a profound impact on search results in this space. Pavel Pudil and Jana Novovicova [1998] have proposed methods for reducing the dimensionality of the problem based on a user level of a priori knowledge regarding the data domain. The authors go on to develop methods for approximation and classification based on probability density functions and feature selection. The second approach uses a divergence method for selecting the most useful features for describing the differences between two possible classes. Their methods would be appropriate to the work presented here if the sole objective was similar part clustering.

The significant issue is that simple Boolean search criteria return rectilinear regions in their respective search space. The "corners" of these rectilinear regions can become quite remote from the region's central point as the dimensionality of the space increases. Thus, there is a high probability that neighbors, as defined by inclusion in these rectilinear regions, will have combinations of part properties that disqualify them as similar parts

even though their individual properties are highly correlated. A more refined search methodology would necessarily take into account relationships between part properties. The most general approach would provide a means of functionally relating every property to all other properties. This in fact is the motivation for "Shaped Neighborhoods" as developed in this paper.

This last notion of relationships between part properties is the thrust of this dissertation. The ability to define and operate on relationships between engineering detail part parameters provides the basis for the "refined search and retrieval" methodology presented in subsequent discussions. These relationships are implemented through a continuum of techniques bounded on one end by a relatively constrained Boolean approach, and on the opposite end by a completely generalized approach involving neural network based shaped neighborhoods in hyperspace. Together, these techniques provide a set of filters or "cookie cutters" which can be stacked (used in series) to provide a search engine capable of operating in million part environments, with the ability to deliver highly refined search results.

2.7 Design Retrieval Methodology

Given the hyperspace discussed above as a vehicle for representing a database of detail parts, the concept of similar parts can now be defined as all parts whose point representations in the defined hyperspace fall within some specified neighborhood of a target part's representation in that space. In classical terms, this specified neighborhood could be taken as the rectilinear region in hyperspace formed around the target part's point representation location (defined here as nominal), plus or minus a specified range of acceptable variability defined individually for each axis (part attribute). For example, a similar part might be a part whose height is equal to the target part's height plus or minus one inch OR whose length is equal to the target parts length plus or minus two inches.

This is characterized as the Union of the two criteria. A slightly more refined criteria is obtained by the Intersection of the two criteria. The various resulting regions are shown below in Figure 2-3.

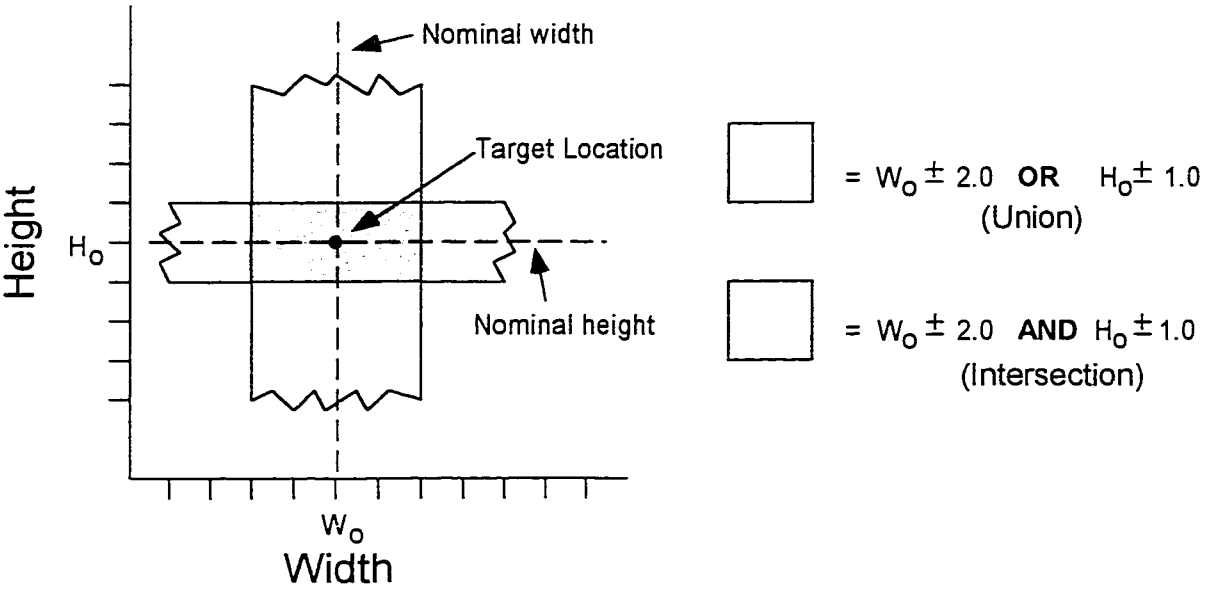


Figure 2-3. Union and Intersection in 2 dimensions.

The above neighborhoods are valid and "similar" parts are discovered. However, the drawback to these classical neighborhood definitions is the fact that the range of acceptable variability on each axis is independent of all other axes. Thus, the relationship of one parameter to another with respect to similarity is not encoded and opportunities for refined matching are missed. A circular neighborhood (in N dimensions) begins to address this issue by limiting the vector formed in hyperspace between a target part and candidate parts in the vicinity to some specified magnitude. However, this approach provides only marginal improvement over the rectilinear neighborhood with respect to property interdependence.

This dissertation extends the above rectilinear and circular definitions of neighborhood to the concept of Shaped Neighborhoods in hyperspace. The ability to functionally relate one parameter to another, indeed all parameters to all other parameters, independently throughout their range, using either analytic methods, a neural network, or a fuzzy logic based approach to neighborhood representation, provides the optimum environment for explicit property specification. This ability leads to highly refined matching in the hyperspace search paradigm. These shaped neighborhoods (analytic, neural network and fuzzy logic based) are referred to as "cookie cutters" in this dissertation for obvious reasons, and they provide a set of interchangeable filters which can be used independently, or stacked, to yield highly refined search results over the detail part hypercube representation. A continuum of 2 dimensional cookie cutters implemented as neural networks are shown in Figure 2-4.

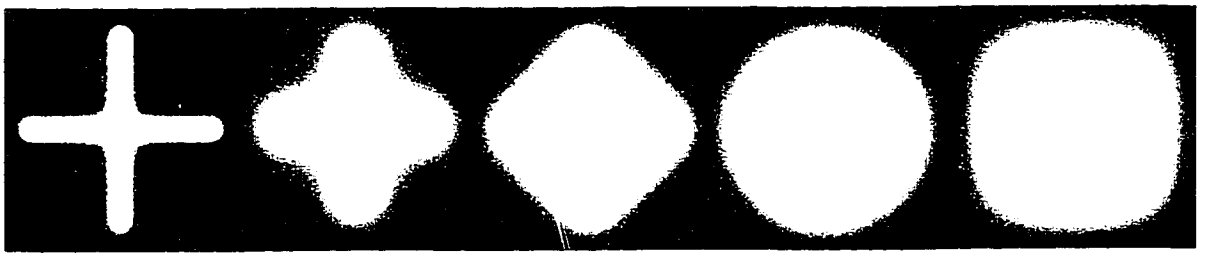


Figure 2-4. A continuum of Neural Network based, 2-D cookie cutter filters.

These particular "cookie cutter" filters are characterized as symmetric as well as analytic (in that they also have a closed form representation). They were derived from an $\mathbf{r}_i^n$ composition where $\mathbf{r}_i$ is the distance from any given point in the domain to its projection on the $\mathbf{i}^{th}$ axis, and $\mathbf{n}$ varies from -2 to +3 going from left to right. The interesting feature here is that this continuum of filters is bounded on the right by the classical rectilinear results discussed above, and bounded on the left by an appropriate

union of SQL queries. The improvements in search capability obtained from the methods derive precisely from the fact that there is a continuum of filters between the bounding extremes, and refined inter-parameter relationships can be defined utilizing this continuum.

For instance, the left most filter defines a high degree of inter-parameter dependence. As one parameter is allowed to vary to the extremes of its range, the second parameter is constrained closer and closer to its nominal value. Conversely, in the right most filter, the two parameters can vary to their respective extremes independent of each other. This right most filter is exactly the same "box" depicted in the classical union relation shown in Figure 2-3. The concept of these 2D shaped neighborhoods extends directly to 3D and above. Figure 2-5 shows a neural network based 3D filter. This image was rendered by an auxiliary ray-tracing algorithm (defined completely in chapter 6) where the neural network supplied filter boundary intersections and normals.

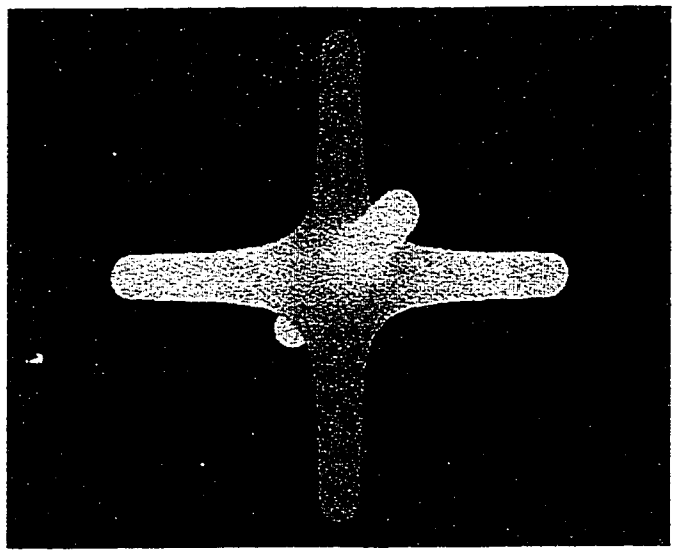


Figure 2-5. Three-dimensional neural network based cookie cutter filter. The image was rendered through a ray-tracing algorithm where the NN supplied intersections and surface normals for given points in space. In actual use, the NN supplies "in or out" criteria corresponding to selected or not selected for a particular search.

Other non-analytic, non-symmetric filters are available in neural net or fuzzy logic form providing infinite variations on search criteria. These filters can be stacked together or used individually to provide a refined search criteria over the hypercube environment as shown in Figure 2-6. Like any series operators, effects at any stage depend on results obtained form previous stages. In this manner, a reasonable number of candidate similar parts can be obtained from among millions, for suitability analysis by a design engineer.

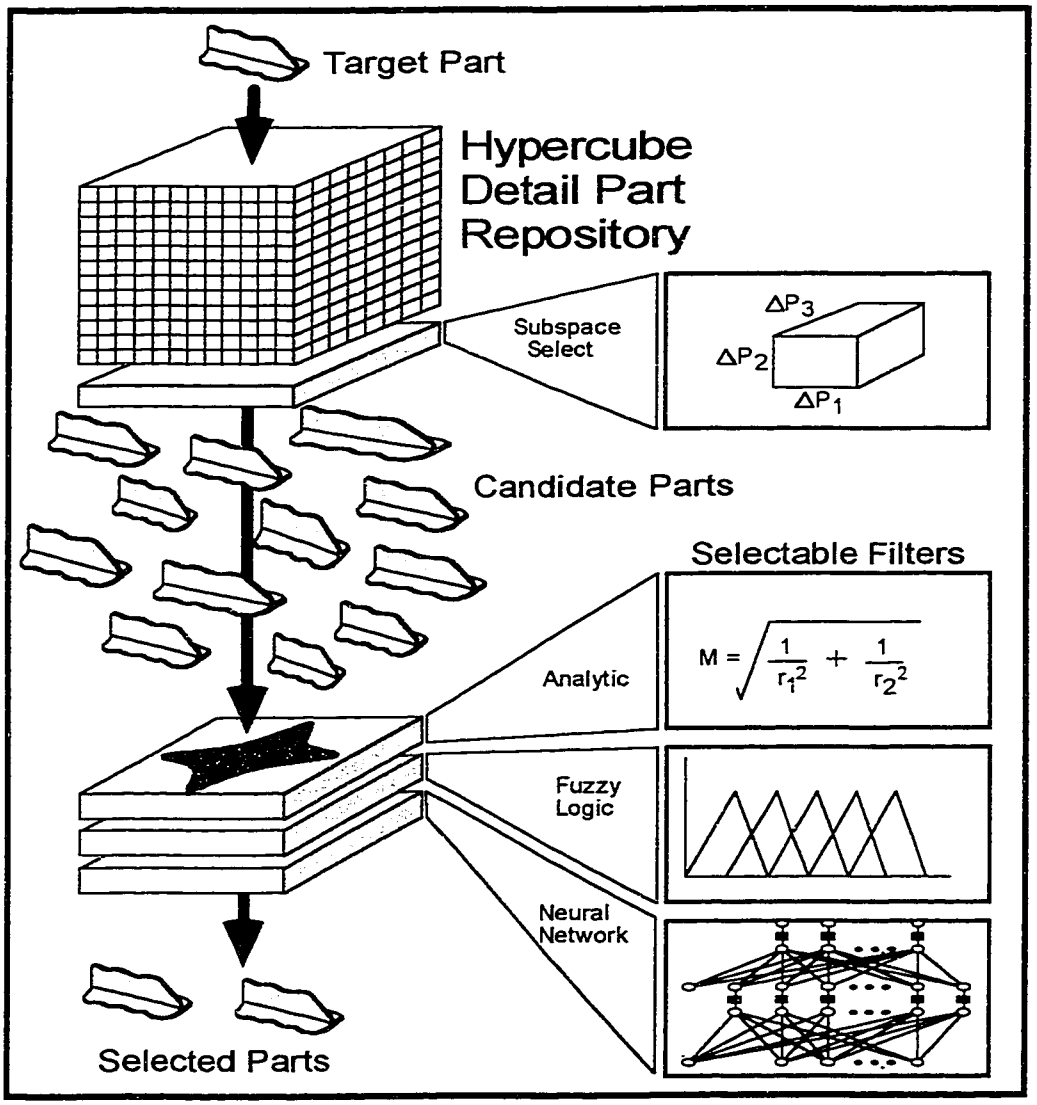


Figure 2-6. Refined search criteria utilizing stacked filters in hyperspace.

The design retrieval system defined in this paper was prototyped on an Enterprise 3000 workstation. The hypercube itself was implemented as a single table in UDB (Universal Database) running on this platform. This is the IBM DB2 relational database running in the workstation environment, supporting an SQL (Standard Query Language) interface. The filters were developed in C programming language.

Chapter 3
Analytic Based Neighborhoods

3.1 Introduction

The first class of filters, and the easiest to implement, are the analytic filters. These filters are indicated when detail part inter-property relationships are easily defined through some analytic expression. An example might be where part weight is required to be proportional to part length for instance. In this case, a target part would establish the constant of proportionality and thereby determine a line through the hypercube. Similar parts would then be selected by virtue of their proximity to this line.

In theory, this sounds fairly straight forward, and in fact it is for very simple relationships. Many times the relationship, including parameter variability, can be established directly in the SQL query itself. Unfortunately, an SQL query based method alone has serious shortcomings and a more generalized approach of analytic filters as shaped neighborhoods post SQL is appropriate. This chapter defines such analytic based shaped neighborhoods, and ultimately sets the ground work for the neural network based neighborhoods of chapter 5.

3.2 Relationships through SQL queries

The simple relationship suggested in section 3.1 above can in fact be established in an SQL query since expressions in SQL can be constructed combining the standard arithmetic operators add, subtract, multiply, and divide. Unfortunately, even with this

simple example, the notion of similarity becomes awkward. For instance, consider just 2 parameters P1 and P2 related through a proportionality constant M and a range of variability R. This basic proportion is shown in equation (7) and the complete relationship is shown in equation (8).

$$P2 = M * P1 \tag{7}$$

$$P1 = [(1/M) * P2] \pm R \tag{8}$$

Then the following SQL would be appropriate:

```
SELECT P1, P2 FROM PARMS.TABLE
WHERE P1 BETWEEN ((P1*(1/M))-R) AND ((P1*(1/M))+R)
```

This query does in fact return a set of database records fulfilling the required relationship. This is shown graphically in Figure 3-1. In this figure, the diagonal line represents equation (7) above and the shaded area represents the region generated by equation (8) as P2 is allowed to vary over some range $P2_0$ to $P2_1$. Probably the most interesting thing to note about this example is that the region carved out by the relationship does in fact have a width related to the required parameter R as indicated by the horizontal lines in the figure, yet this region is not generated by the perpendicular offset as might reasonable be expected. In other words, the width of the region R would also have to be expressed in terms of the proportionality constant M. The bottom line is that while simple analytic relationships can be established in SQL alone, they are limited and it is more reasonable to take the more useful aspects of the analytic relationship and transfer it to a shaped neighborhood definition. This transformed analytic definition can now be use it exactly the same way the fuzzy logic and neural net filters are used.

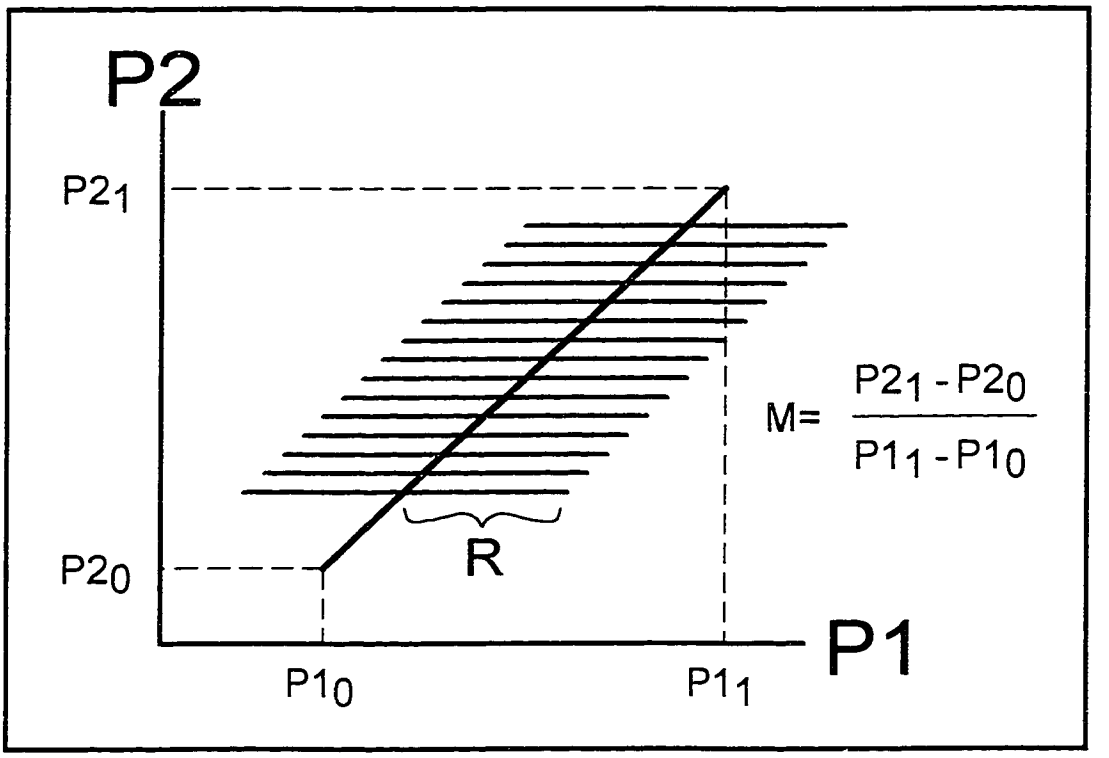


Figure 3-1. P1 and P2 related through an SQL query.

3.3 Generalized Analytic Filters.

As was suggested in section 3.2, it is reasonable to develop a shaped neighborhood approach to analytic based filters, and implement them the same as fuzzy logic and neural network based neighborhoods with respect to similar part discovery. A generalized analytic approach is facilitated through a set of filters based on the ℓ_α-norm. The idea here is that given a set of N-dimensional axes (representing a detail part parameterization), basic standard relationships can be written analytically between these parameters. These relationships carve out regions in space which become the discriminator for selecting similar parts. A target part establishes an origin in the space, and the analytic relationship

defines a region around that target location. Similar parts to the target part lay within this region. . The ℓ_1-norm, ℓ_2-norm, and ℓ_∞-norm which are the basis for a class of analytic filters are defined in Figure 3-2.

$$\ell_1\text{-Norm}: \quad \| x \|_1 = \sum_{r=1}^{n} | x_r |$$

$$\ell_2\text{-Norm}: \quad \| x \|_2 = \left(\sum_{r=1}^{n} | x_r |^2 \right)^{1/2}$$

$$\ell_\infty\text{-Norm}: \quad \| x \|_\infty = \max_i | x_i |$$

Figure 3-2. The ℓ_1-norm, ℓ_2-norm, and ℓ_∞-norm

More specifically, a vector field is defined over the space, where the vector magnitude is proportional to the L_α-norm (where α is real valued) of the point and its projections on the orthogonal axes defining the space. The vector direction is normal to the iso-surface interpolated by points of equal vector magnitude. Figure 3-3 shows a two dimensional example of a family of neighborhoods generated by this approach. In this example, a target part defines a location (H',W') in the hyperspace. Choosing α=-2.0 defines a highly dependent neighborhood where only one parameter is allowed to range to its extreme at a time, thereby constraining the other parameter near its nominal value. Choosing α=-1.4 somewhat relaxes this constraint.

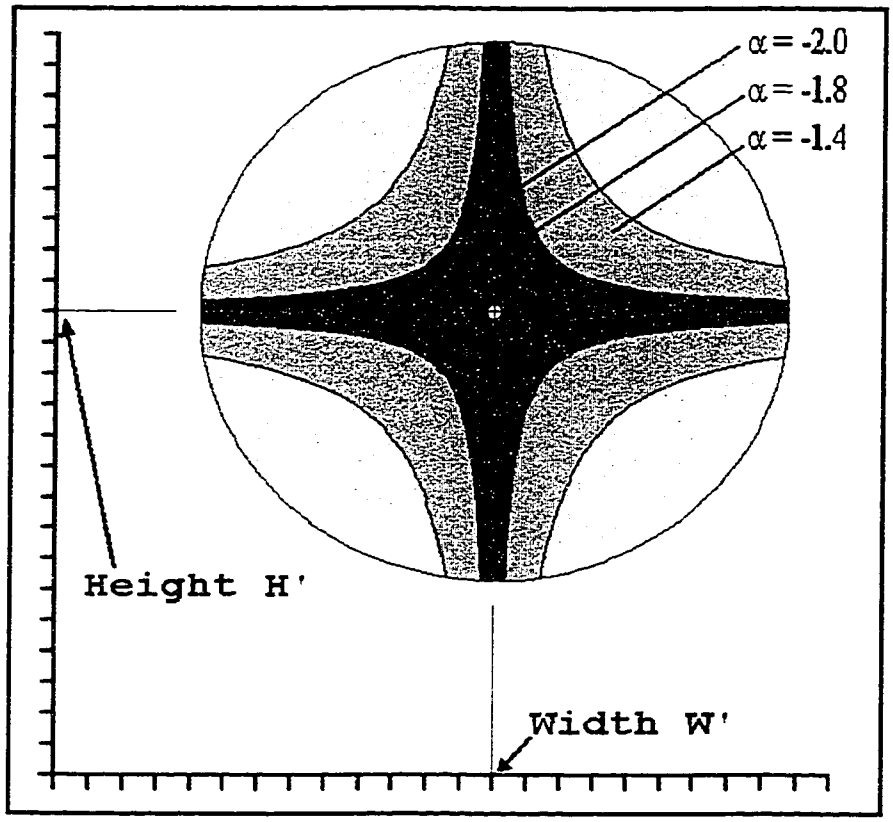


Figure 3-3. A family of two dimensional neighborhoods generated by the L_α-norm.

This analytic class of filters is appropriate as a first approximation to search refinement. As α decreases from +3.0 to -2.0, more and more emphasis is placed on parameter inter-dependence. At α = -2.0, an individual parameter may range to its extreme only by forcing all to other parameters to approach their nominal values.

L_α-norm analytic based shaped neighborhoods provide a useful class of filters. There are other analytic expressions which can be made the foundation for other equally useful shaped neighborhood filters in the context of similar part discovery. Analytic filters also set the stage for the fuzzy logic filters of chapter 4 and the neural network based filters of chapter 5.

Chapter 4
Fuzzy Logic Based Neighborhoods

4.1 Introduction

A second class of database search filters was developed around a fuzzy logic methodology. In particular, two distinct fuzzy logic approaches to similar detail part discovery and selection were studied and implemented. The first approach essentially involves performing a detail part fuzzy comparison parameter by parameter to build up a fuzzy figure of merit which is then interpreted to provide part discrimination. This is somewhat analogous to the straight analytic approach of section 3.2 in the sense that a specific comparison between two parts is evaluated with disregard for any generalized description of similarity.

Following this analogy, the second approach is in fact a generalized neighborhood approach based on fuzzy logic. In this case, ranges of parameter variability are assigned fuzzy membership and a heuristics table provides an antecedent/consequence response ultimately leading to generalized fuzzy logic based shaped neighborhood. These neighborhoods are then used to provide detail part discrimination in exactly the same way as the analytic methods of chapter 3, namely by bounding a region in space in a sort of "in or out" paradigm. Points (which represent instances of detail parts in the space) which fall inside the region boundary are considered similar to a target part (located at the origin of the bounding region) and become candidates for part reuse. This chapter develops both methods and concludes with a fuzzy logic to analytic filter comparison.

4.2 Parametric fuzzy logic comparison

The objective of the fuzzy comparison method described here is to provide a means of determining to what degree a candidate part might be considered similar to a target part based on their respective part properties. The methodology provides a means of quantifying similarity while providing the notion of fuzzy matching where it is not necessary to explicitly state match criteria.

As with classical fuzzy systems, the input (part properties) are first fuzzified. Next, an antecedent/consequence analysis is performed and finally, the results are de-fuzzified to produce a crisp output, in this case, a measure of part similarity as shown in Figure 4-1.

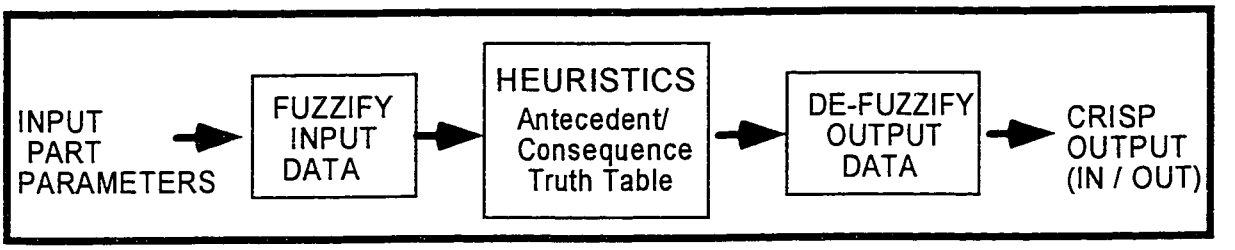


Figure 4-1. Overview of fuzzy logic filter.

In particular, both target and candidate part properties are fuzzified using the membership function shown in Figure 4.2. Each property (P1 through P8) is assigned membership in the five classes by virtue of its magnitude. This membership ranges from 0.0 for not in the group to 1.0 for completely belonging to the group. As was mentioned in the introduction to this chapter, while this approach does provide similar part discovery, the method is considered somewhat limited. This is due to the nature of the fuzzy matching, namely, while individual parameters do match up, a means of characterizing general parameter relationships is not provided.

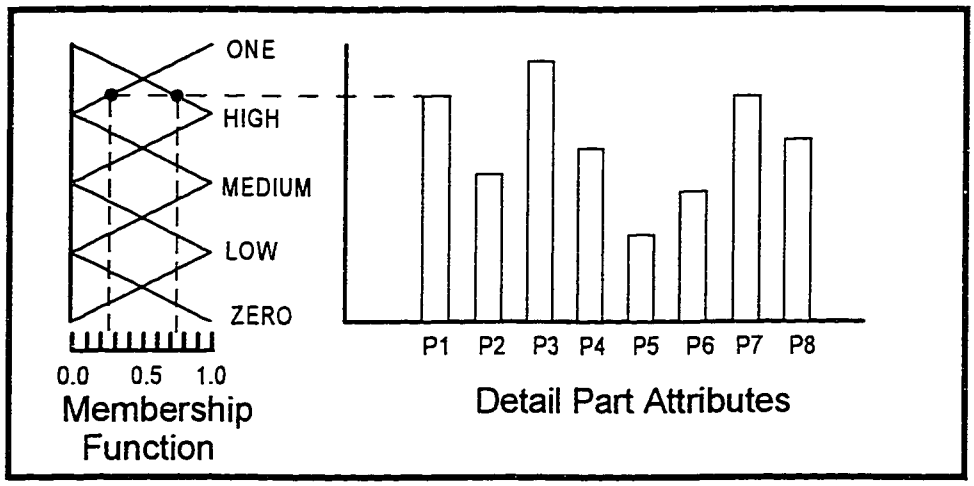


Figure 4-2. Fuzzy membership function for part properties.

Next, a fuzzy rule table provides the detail part comparison heuristics as shown in Figure 4-3. For instance, the following heuristic is explicit in the table:

If TARGET IS HIGH AND CANDIDATE IS MED
THEN CONSEQUENCE IS POS

For example, if a target part parameter 1 belongs to HIGH with membership 0.7 and MED with membership 0.3 and a candidate part parameter 1 belongs to MED with membership 0.4 and LOW with membership 0.6, then from the table the consequence membership is determined as VPOS=.3, POS=0.3+0.4=0.7, ZE=0.6. Note that due to the nature of the fuzzy membership function, exactly 4 adjacent squares will be active for any given comparison, and using min/max interpretation, a given square will be assigned the minimum of the two defining values.

VPOS = min (candidate(MED), target(MED))
= min (.4, .3)
= .3

CANDIDATE PART

TARGET PART	ONE	HIGH	MED	LOW	ZERO
ONE	VPOS	POS	ZE	NEG	VNEG
HIGH	POS	VPOS	POS	ZE	NEG
MED	ZE	POS	VPOS	POS	ZE
LOW	NEG	ZE	POS	VPOS	POS
ZERO	VNEG	NEG	ZE	POS	VPOS

Figure 4-3. Antecedent/Consequence table.

This is repeated for each parameter and the results are collected through addition. Finally, the summed results are de-fuzzified using a central tendency approach as shown in Figure 4-4. The fuzzy filters described above provide an opportunity for introducing flexibility (fuzziness) in selecting similar part while maintaining a refinement over the rectilinear regions described above. This is particularly effective in this application in limiting the number of selected parts returned while allowing certain parameters to be far ranging with respect to other part parameters, especially when the relationship between those parameters is nor necessarily known.

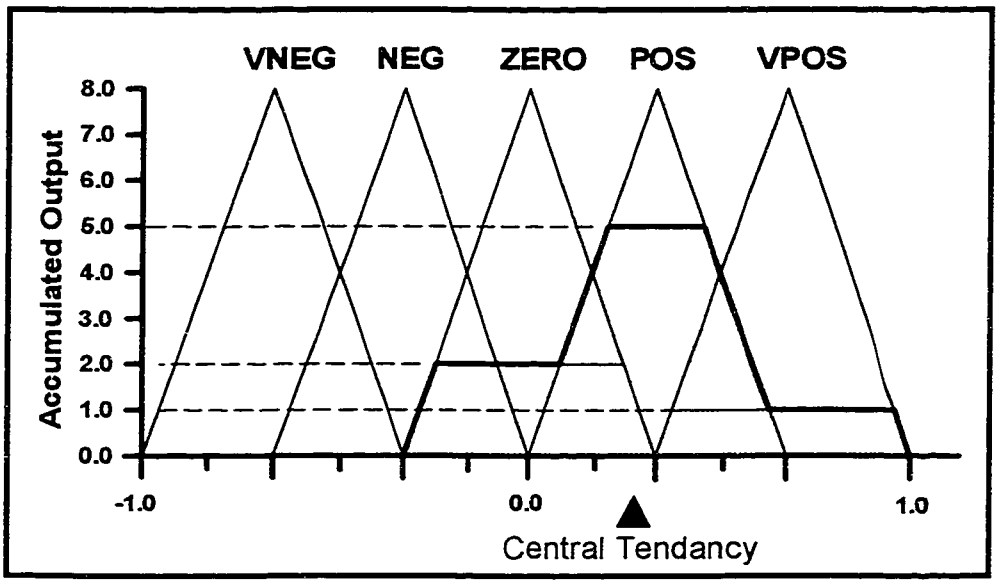


Figure 4-4. Defuzzification using central tendency.

As can be seen from Figure 4-4, in the example given, the final de-fuzzified figure of merit for the particular comparison is approximately 0.27. This would be interpreted as slightly less than positive and subjectively would not correspond to a very good match.

4.3 Generalized fuzzy logic based shaped neighborhoods

A more useful and generalized approach to fuzzy logic based detail part selection is through the use of generalized fuzzy logic based shaped neighborhoods. Such neighborhoods are completely compatible with the analytic neighborhood filters defined in chapter 3. These fuzzy logic based shaped neighborhoods are considered generalized because each parameter can be functionally related to all other parameters throughout their entire range. In this case, the relationships are explicitly defined in the antecedent/consequence table.

Specifically, detail part parameters are fuzzified over their range using classical triangular fuzzy membership functions. In this manner, a given neighborhood location corresponds to a set of fuzzy membership values with respect to each of the 8 coordinate axes as shown in Figure 4-5.

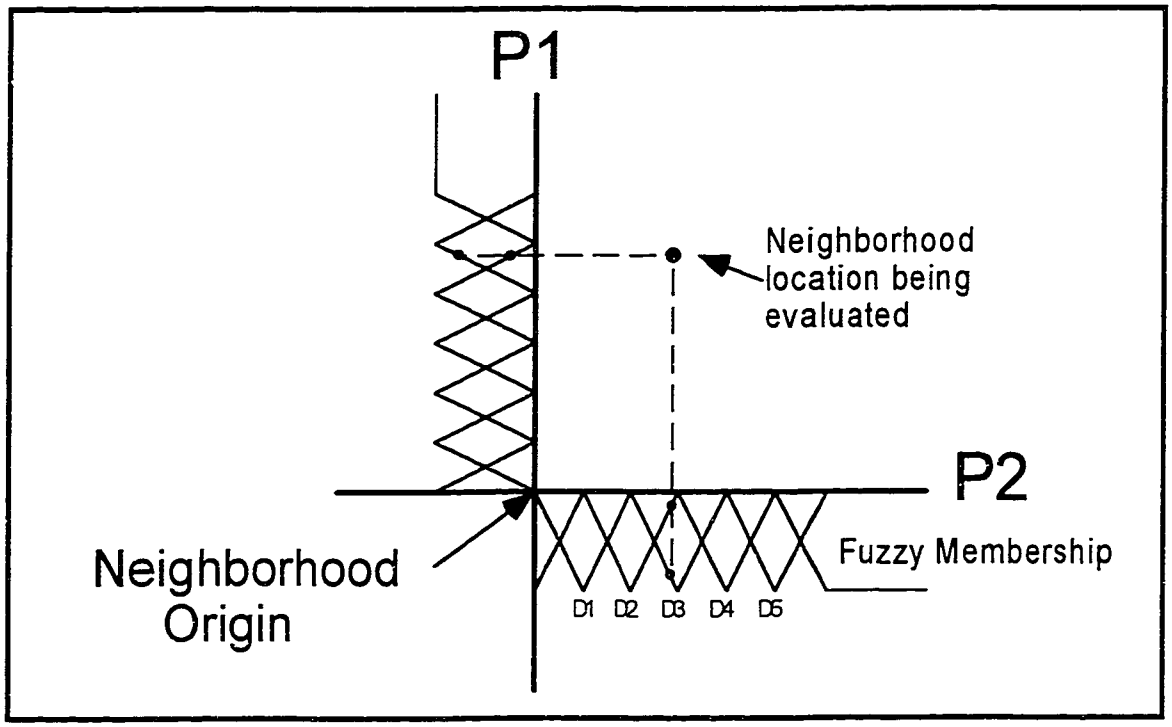


Figure 4-5. Fuzzy membership functions for neighborhood locations

An 8 dimensional decision surface table establish the relationship between various parameters which ultimately defines the shape of the resulting neighborhood. Qualitatively, this table expresses the idea that nearness to the neighborhood origin corresponds to strength of similarity in the de-fuzzified output. A single plane of this table for a typical example is shown in Figure 4-6. Note that the table essentially exposes the inter-parameter relationships through the value associated with each table node.

Parameter 2	Parameter 1					
	D1	D2	D3	D4	D5	D6
D1	9	6	4	3	2	1
D2	6	5	4	2	0	0
D3	4	4	3	1	0	0
D4	3	2	1	0	0	0
D5	2	0	0	0	0	0
D6	1	0	0	0	0	0

Figure 4-6. Parameter relationships expressed as a decision surface.

This table was used to generate an image of a fuzzy logic based shaped neighborhood as shown in Figure 4-7. (This was accomplished by evaluating all neighborhood locations around an origin as indicated in Figure 4-5 above.) The strength of the de-fuzzified figure of merit was used to encode the pixel intensity of figure 4-7. This image is interpreted as showing the fuzzy relationship between two parameters (the X and Y axis respectively) by indicating the strength of the relationship (pixel intensity) at all points in the region. The white area thus defined indicates a region whose corresponding parameters meet the fuzzy criteria of similarity to a hypothetical point located at the origin. This particular shape is strongly correlated to the l-alpha analytic neighborhoods of chapter 3.

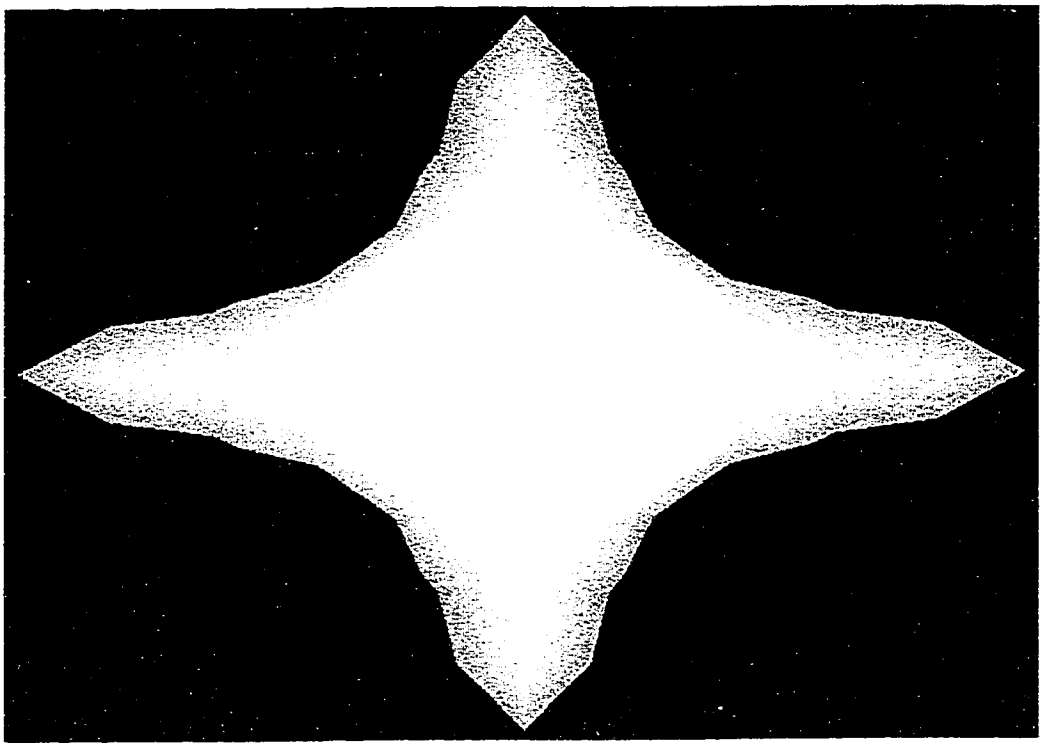


Figure 4-7. Fuzzy Logic based shaped neighborhood.

4.4 A fuzzy logic shaped neighborhood case example

The methods defined in the section 4.3 above were used to select similar detail part candidates from a subset of 2000 parts meeting a minimum rectilinear criteria. Figure 4-8 shows 2 parts that were determined similar based on these methods while figure 4-9 shows a part that was rejected. Note that the bounding box for these three parts is nearly identical and discrimination is therefore accomplished against surface area and volume.

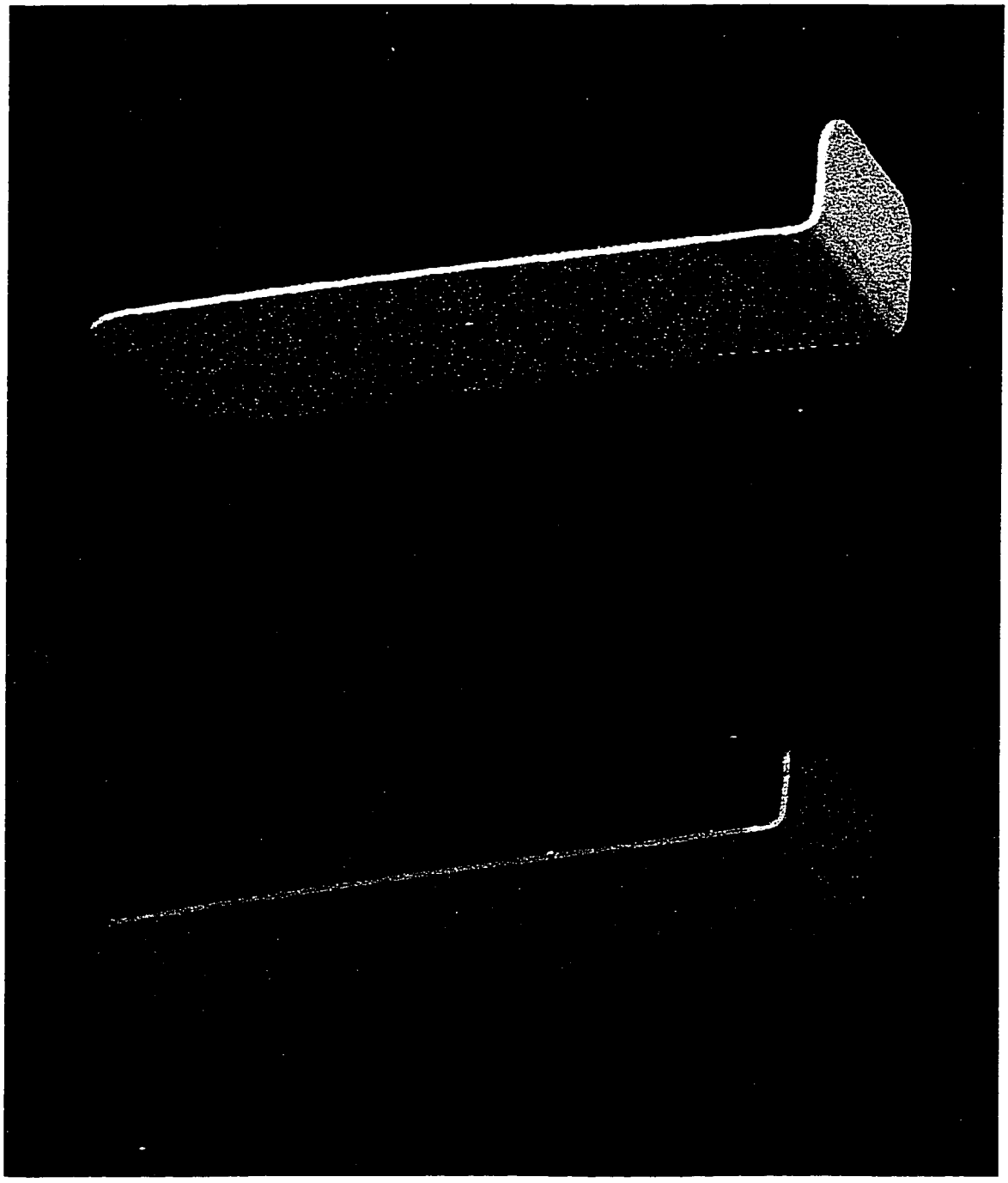


Figure 4-8. Two similar part detected with fuzzy neighborhoods. Top part is slightly longer that bottom part.

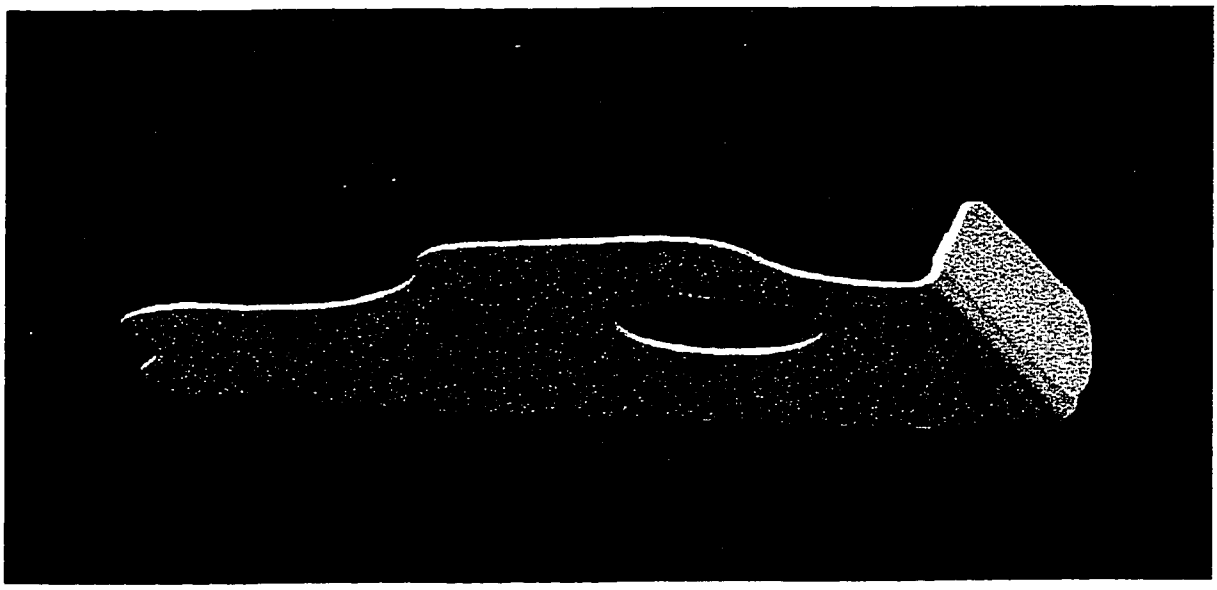


Figure 4-9. A rejected part not within the fuzzy neighborhood.

The fuzzy logic methods presented in this chapter do provide an excellent environment for evaluating and selecting similar detail parts given a target part objective. The fuzzy neighborhoods are useful to work with and neighborhood expansion is readily available by simple interpretation of the de-fuzzified output figure of merit. In this manner, an overall similarity sensitivity is established and a wide variety of useful results can be obtained.

The fuzzy methods of this chapter provide an interesting counterpoint to the analytic methods of chapter 3. In the next chapter, neural network based shaped neighborhoods are developed leading to an interesting equivalence between all 3 methods. It is this equivalence that provides the utility of the overall approach to similar part detection presented in this dissertation. An environment supporting inter-changeable filters, each with their respective strengths, provides the desired generalized search functionality.

Chapter 5
Neural Network Based Neighborhoods

5.1 Introduction

This chapter continues with the theme of shaped neighborhoods in 8 dimensional space as a vehicle for selecting similar engineer detail parts from a subspace of pre-selected part instances. The space contains instances of individual detail parts represented as points in the space, where the coordinates of the points are the individual part property values (length, surface area, etc.). Instances of detail parts in the subspace are identified through classical SQL queries against the entire space, the latter being implemented as a DB2 relational data model on a Universal Database (UDB) hosted on a Sun workstation on top of a Solaris operating system.

Previous chapters have developed the notion of shaped neighborhoods in this space based on analytic and fuzzy logic approaches. These shaped neighborhoods are centered on a target part representation in the space, and thereby form a neighborhood region in the subspace. Part instances bounded by these refined regions are considered similar parts to the target part and are potential reuse candidates. This chapter goes on to present shaped neighborhoods based on a NN approach. It will be shown that these neighborhoods are compatible with and complement neighborhoods previous defined using analytic and fuzzy logic approaches. The neighborhoods act as filters and can be employed singly or as in a series.

5.2 Neural Network Topology

The NN used to implement the cookie cutters defined in this dissertation is shown in Figure 5-1. It is a classical layered perceptron with 8 input nodes corresponding to the 8 hyperspace axes, a single hidden layer, and 8 output nodes which define the 8 components of the vector field spanning the hyperspace domain. The output layer is fitted with a magnitude filter defined in terms of the L_α-norm which ultimately determines the cookie cutter shape.

Conceptually, the neural network based cookie cutter is placed in the hyperspace at the desired target coordinates thereby defining the neighborhood of similar objects. In practice, since the cookie cutters are all trained at the origin, the hyperspace is actually brought to the cookie cutter. This is a quick and efficient offset process which takes nothing away from performance. A distinction must be made at this point with respect to classical search heuristics. Mihalis Yannakakis [1990] characterizes various classes of search problems as optimization over a set of solutions with respect to some cost function. Search heuristics are obtained by superimposing a neighborhood structure on a set of solutions and iterating towards better and better neighboring solutions. This is contrasted to the notion of neighborhood presented here where the domain is considered static and the shaped neighborhood boundary definitions have been predetermined.

The use of a neural network representation also provides a means of defining a fuzzy membership with respect to similar parts. While it is true that a given alpha surface (locus of equal vector magnitude) defines a crisp boundary on the neighborhood, the network does return a value for parts outside the neighborhood which can be interpreted as a fuzzy membership value for that particular object. This is important in that it provides a direct method of implementing "degree of similarity" in the shaped neighborhood approach.

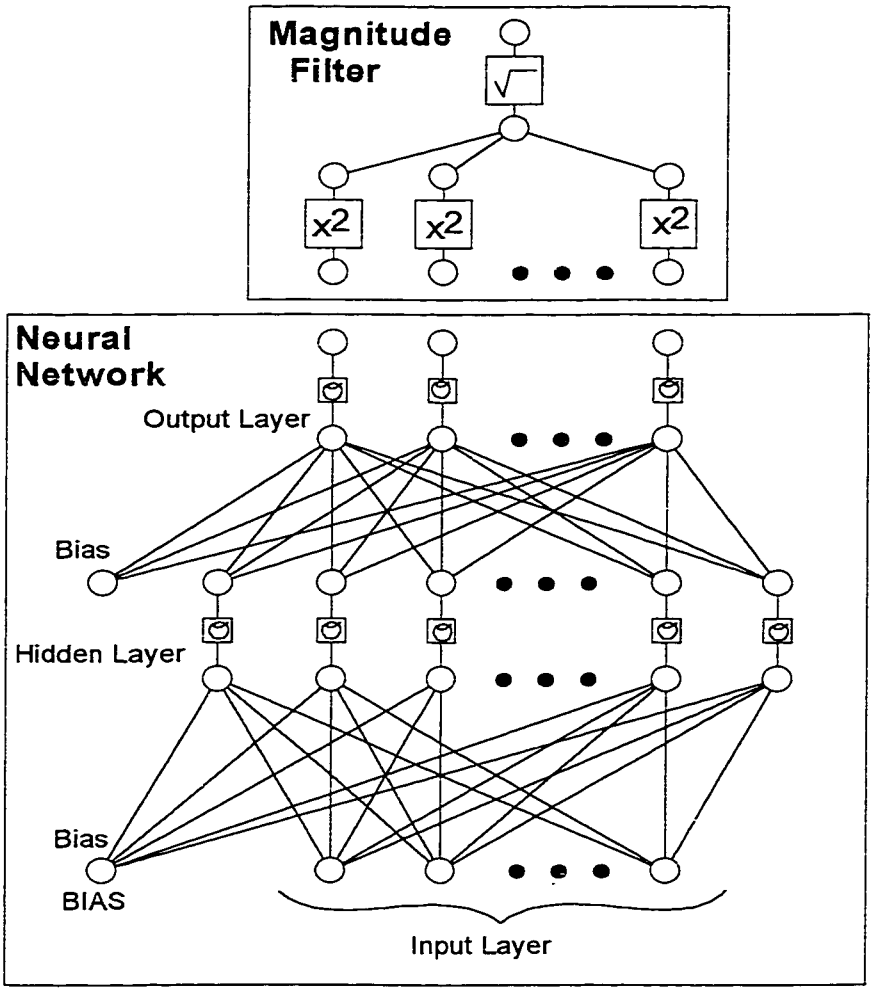


Figure 5-1. Neural Network topology used to define cookie cutters. The σ denotes the classical Sigmoid non-linearity. Note that the output layer defines a vector field where every point in the input space is associated with a vector whose magnitude and direction define the shaped neighborhood.

Detail part attributes including mass properties, non-graphic attributes, and a thumbnail image are stored in a single relational database table. A typical search scenario includes search parameter specification and appropriate filter selection. The parameter specification fixes a sub-space in the hypercube thereby defining the rectilinear boundaries enclosing the candidate parts. These are retrieved through the database using

standard SQL. Next, this list of candidates (potentially numbering in the thousands) is passed through one or more filters (cookie cutters) yielding refined search results. The filter stack can easily and quickly be manipulated to reduce the number of selected parts from hundreds to a reasonable number, say ten, for human inspection. The thumbnail image provides efficient visualization for this purpose.

5.3 Network Training

The neural network based shaped networks presented here have an added advantage over the analytic neighborhoods presented in chapter 3 in that they are directly adaptable to expert opinion. This is evident in one method of obtaining network training data. Specifically, training data is produced by programmatically monitoring an expert design engineer in the process of manually selecting similar part from a particular part type, shear ties for instance. All parts examined by the expert become training instances where the input vector is set to the individual part parameterization (length, width, volume, etc.) and the output target set is set to 1 for selected parts and 0 for rejected parts. Thus the expert provides input relative to their understanding which is then captured by the NN.

A standard network topology as shown in Figure 5-1 is trained on this dataset and the resulting weight are stored as a shaped neighbor (cookie cutter) associated with the particular part type. Neighborhoods are trained once and used many times. A collection of cookie cutter is built up to assist design engineers in discovering similar detail parts appropriate to their current requirements, for potential part reuse. Cookie cutter portability derives from the fact that a standard network topology is used and transporting neighborhoods between tools and applications is accomplished by simply moving an array of 143 real values around.

5.4 Shaped Neighborhood Results

The neural network shaped neighborhoods were used to select similar parts from a subspace of parts using a generalized neighborhood filter. In subjective terms, this filter promotes the notion of constraining variability by allowing only one part property to range to its extreme at a time. As a given property diverges from its nominal value, all other properties are constrained closer and closer to their respective nominal values. The system performed well on both scale and selectivity factors as defined above. The fact that the working environment is the hypercube topology where data instances are represented as single points yields a highly efficient system which can be easily "loaded" with millions of parts. Eight dimensional space is quite "roomy".

This system was bench-marked against a set of shear tie detail parts. The hypercube was populated by 1200 plus instances of detail parts from a number of related families in order to approximate a typical space density for the trials. This implies that much of the hypercube domain was not populated. Even so, it is reasonable to assume that results obtained for the shear ties associated with this region could be generalized to other regions of the space representing other part types and families. A neural network filter relating length to weight was defined and used as the selection criteria for the trial. By choosing the part instances from related families, the individual parts were already quite similar, thus the 1200 parts could be considered as representing a typical selected set as returned by a classical rectilinear search algorithm. The system was able to further discriminate between various part characteristics through the use of shaped neighborhood as defined by the neural network reducing the 1200 parts to 6 candidates as shown in Figure 15-2.

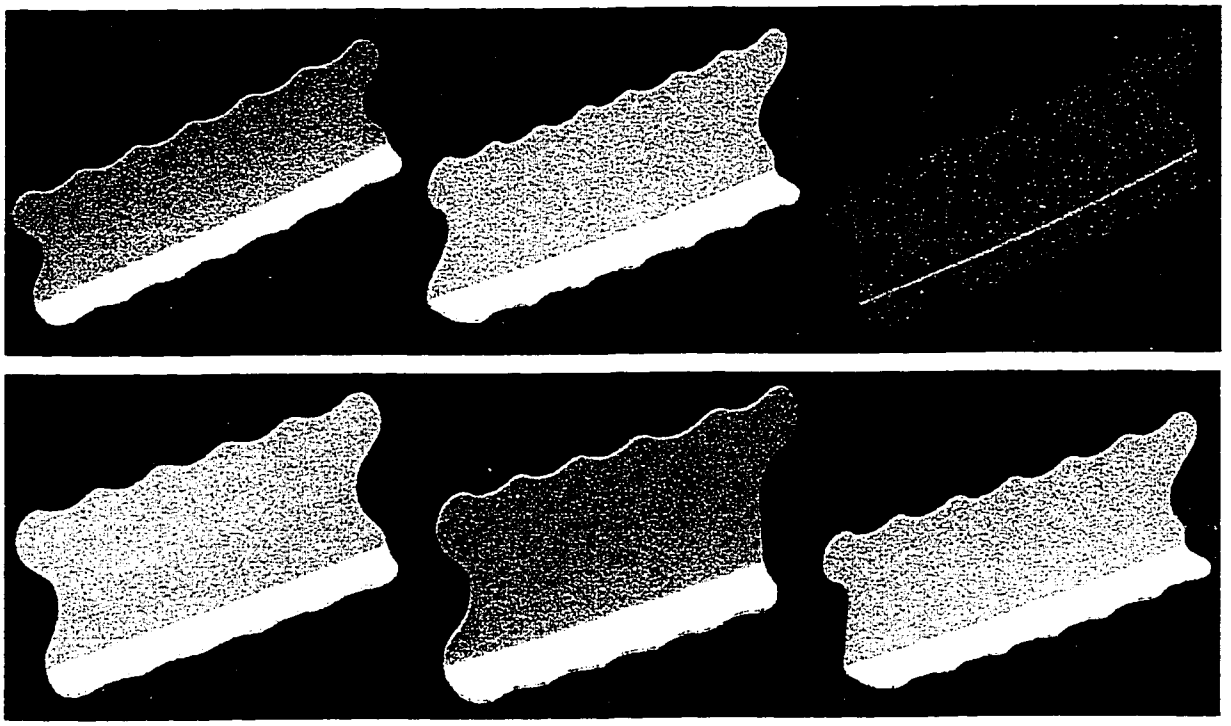


Figure 5-2. The 6 parts selected by the neural network shaped neighborhood.

The majority of the 1200 parts were not selected by the filter for various reasons. More specifically, while these parts had individual mass properties similar to the target part, they did in fact have combinations of parameters which placed them outside the range of the scope of the filter and were consequently not selected. Three typical parts not selected by the neural network based shaped neighborhood filter are shown in figure 5-3.

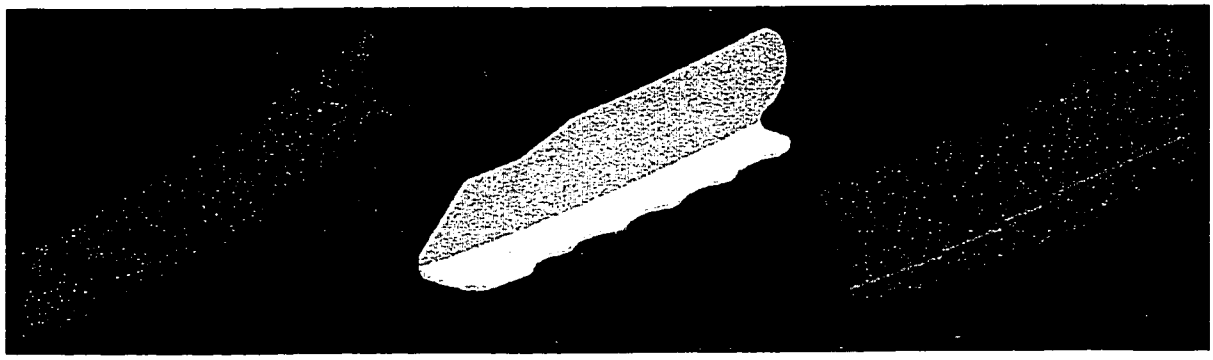


Figure 5-3. Three of six parts passing through the SQL rectilinear filter yet ultimately being rejected by the neural net based "cookie cutter".

Neural network based shaped neighborhoods provide a completely generalized environment for defining relationships between engineering detail part properties. These relationships are exploited in this environment to yield a method of discovering similar parts in a design retrieval system. The neural network provides the capability of expressing these relationships as a functional representation over the domain. This is due to the ability of a neural network to capturing these functional relationships. Further, these functions typically do not have a reasonable analytic, closed form expression which would yield to other less exotic search methods.

It is useful for understanding the role of NN based shaped neighborhoods by considering a range of search approaches in terms of their ability exploit inter-parameter relationships. This range is bounded on one extreme by SQL type searches with essentially no inter-parameter relationship capability. In the middle of the range are analytic type inter-parameter characterizations. On the other extreme bound are the completely generalized NN based shaped neighborhoods where 8 separate parameters can be inter-related concurrently.

Chapter 6
Neighborhood Visualization Methods

6.1 Introduction

A number of software tools and techniques were developed in support of shaped neighborhoods as presented in this dissertation. An optimized neural network topology was established in chapter 5, and this topology was implemented in code in a modular fashion such that the working network could be treated as a black box with a set of standard inputs and outputs, plus bi-directional access to all network weights at all times. This modular approach facilitated the various outboard tools and methods presented in this dissertation. The modular approach is shown in Figure 6-1.

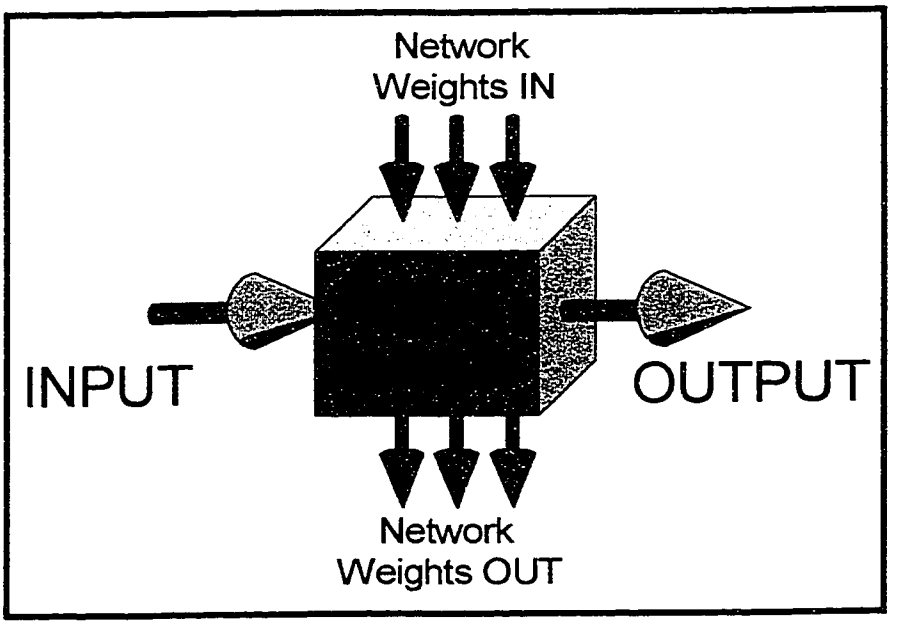


Figure 6-1. Modular, black box approach to neural network implementation.

The modular approach shown facilitates a number of useful capabilities. First, the approach provides complete portability of trained network results between the various tools and applications incorporating the network. Without belaboring the point, suffice it to say that given equivalent network topologies throughout the various applications (which is the working premise of this dissertation), portability of training results is synonymous with the ability to convey the network weights as a set of real numbers between these application. Second, networks can be simply plugged into applications, and as long as the applications stay within the input domain, results are guaranteed. Finally, the real time access to the network weights provides flexibility with respect to controlled initialization of weights during training. Intermediate training results are captures at the Network Weights out node and feed back in later thus providing a "save and restore" network capability.

This chapter presents a visualization method developed around the neural network implementation shown above. The code implementation is characteristic of all the tools and applications developed in this dissertation, and as such provides useful insights into the methods and techniques. Appendix A presents the complete software program used to render shaped neighborhoods into TIFF files suitable for viewing.

6.2 Neural Network Based Ray Tracing

The neural network based shaped neighborhoods presented are developed around a vector field concept. Specifically, every point in some arbitrary high dimension space has a vector associated with it. The dimension of the vector is equal to the dimension of the space. The magnitude of the vector defines an iso surface interpolating the point and the direction of the vector is normal to this surface at the point.

Thus, the locus of all points whose vector magnitude is some specified value forms a boundary surface in the space. Shaped neighborhoods are bounded by these surfaces.

A software method for visualizing these surfaces involves "casting rays" through the domain with respect to a particular view location called eye point. A schematic of the method is shown in Figure 6-2.

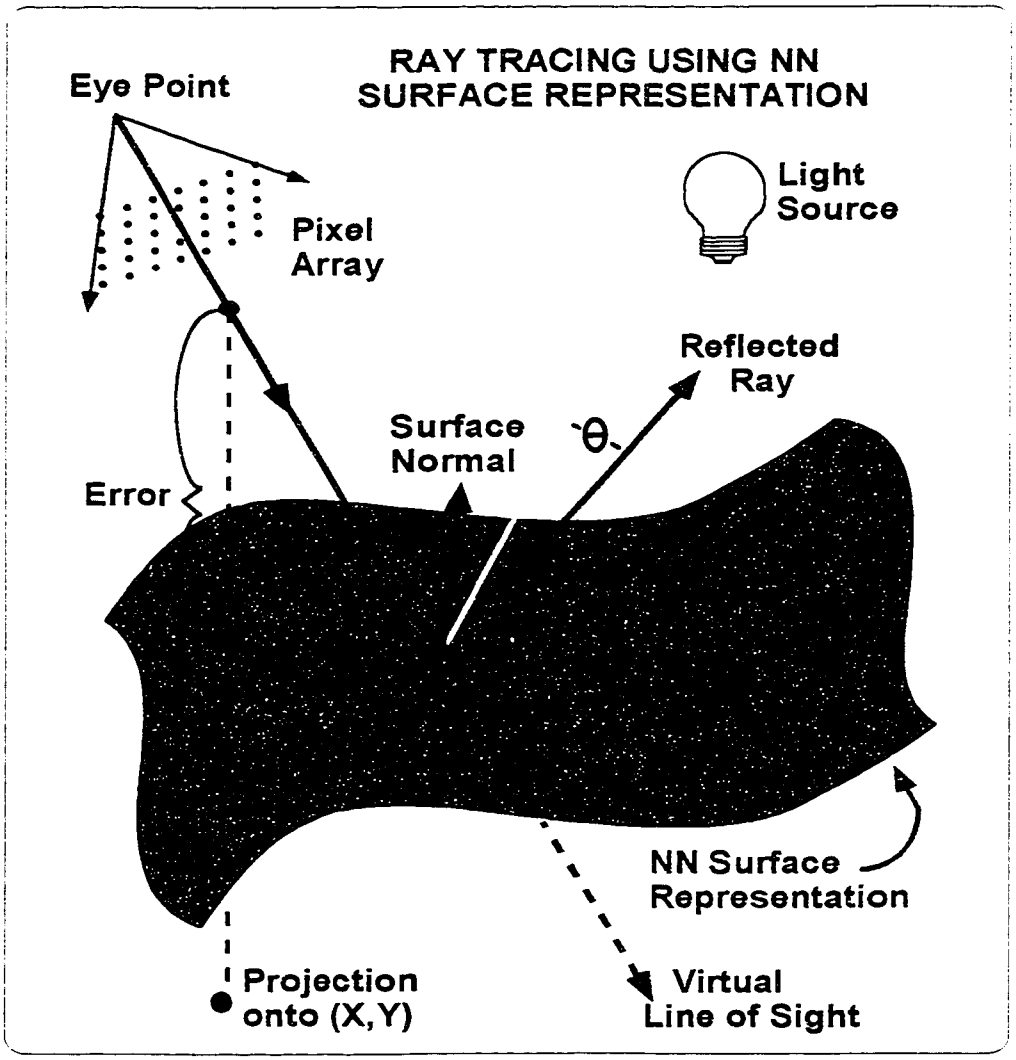


Figure 6-2. Neural network visualization schematic.

The process begins with the selection of a view or **eye point**, and the specification of a **line of sight**. Next, a resolution is chosen and the three parameters together establish a two dimensional **pixel array** in space. This pixel array is normal to the line of sight and is essentially a construction aide for the subsequent casting of rays. Now, individual rays are projected from the eye point through each pixel array point and a test point is evaluated along this ray. In essence, the test point is pushed along the ray until the error (difference between the current NN evaluation and desired evaluation) reaches some minimum value (say 0.001). The test point is now coincident with the desired surface.

Next, the surface normal is determined and the incident ray is reflected around this normal. The angle between the **reflected ray** and some defined **light source** location is proportional to the pixel color (intensity) for that particular ray. The process is repeated for all pixel array points (480 X 640), and a TIFF image is formatted from the results.

6.3 Code analysis

This section examines selection portions of the code written to implement the neural net based shaped neighborhood visualization function. It is instructive to notice how some of the NN visualization features are accomplished. Figure 6-3 shows the declaration portion of the code where various image and network parameters are declared.

```
C DECLARE NEURAL NETWORK ARRAYS AND PARAMETERS
      INTEGER*4   IMAGE(480,640)
      REAL*8      WT1(20,4), WT2(21)
      REAL*8      NODE1(4), NODE2(21), NODE3, SIGPRIME
      REAL*8      ACT1(20), ACT2, ERR, LRP
```

Figure 6-3. Network declarations.

Specifically, these declarations establish the network weight arrays, the network nodes, the sigmoid non-linearity, the activation, and the Learning Rate Parameter (LRP). Next, Figure 6-4 demonstrates the bi-directional network weight read/write capability.

```
C READ IN THE WEIGHTS DATA
8       FORMAT(F32.22)
        DO 12 J=1,20
        DO 10 K=1,4
           READ(UNIT=65,FMT=8,END=20,ERR=20)VAL
           WT1(J,K)=VAL
C          WRITE(6,*)J,K,WT1(J,K)
10      CONTINUE
12      CONTINUE
14      FORMAT('WT1(',I2,',',I2,')=',F32.28)
        DO 16 K=1,21
           READ(UNIT=65,FMT=8,END=20,ERR=20)VAL
           WT2(K)=VAL
C          WRITE(6,*)K,WT2(K)
16      CONTINUE
19      FORMAT('WT2(',I2,')=',F32.28)
        GOTO 100
```

Figure 6-4. The network weights I/O code.

Thus an external file is created containing the weights as real values as shown in Figure 6-5. This provides the portability of trained network results between applications.

```
-3.270221254413615330000
 0.145434798538462570000
-0.406407996454165574000
 1.730593399567526000000
-0.534187134575547909000
-0.560990936831194692000
   ...
```

Figure 6-5. A portion of the network weights available externally.

Finally, the Network evaluation code is presented in Figure 6-6. It is a single feed forward pass through the network to evaluate vector magnitude for a specific input.

```
C*************************************************************
      SUBROUTINE EVAL(ZPNT,MAG)
C*************************************************************
C  EVALUATE THE NEURAL NETWORK AT THE INPUT SPACE POINT
C*************************************************************
IMPLICIT INTEGER*4 (A-Z)
C
C DECLARE NEURAL NETWORK ARRAYS AND PARAMETERS
      REAL*8     ZPNT(3), MAG
      REAL*8     WT1(20,4), WT2(21)
      REAL*8     NODE1(4), NODE2(21), NODE3, SIGPRIME
      REAL*8     ACT1(20), ACT2, ERR, LRP
C
C FEED FORWARD TO NODE 1
      NODE1(1)  = ZPNT(1)
      NODE1(2)  = ZPNT(2)
      NODE1(3)  = ZPNT(3)
      NODE1(4)  = -1.0
      NODE2(21) = -1.0
C
C*****************************************
C FEED FORWARD TO HIDDEN LAYER 1
      DO 150 T=1,20
         ACT1(T)=0.0
         DO 140 U=1,4
            ACT1(T)=ACT1(T)+(NODE1(U)*WT1(T,U))
140      CONTINUE
150   CONTINUE
C     WRITE(6,*)'ACT1=',ACT1
C
C CALCULATE THE NON-LINEARITIES FOR HIDDEN LAYER 1
      DO 160 N=1,20
         CALL SIGMOID(ACT1(N),NODE2(N))
160   CONTINUE
C     WRITE(6,*)'NODE2=',NODE2
C
C*****************************************
C FEED FORWARD TO OUTPUT LAYER
      ACT2=0.0
      DO 170 V=1,21
         ACT2=ACT2+(NODE2(V)*WT2(V))
170   CONTINUE
C     WRITE(6,*)' POINT =',ZPNT
C     WRITE(6,*)' ACT2  =',ACT2
C
C CALCULATE THE NON-LINEARITIES FOR OUTPUT LAYER
      DO 185 N=1,3
         CALL SIGMOID(ACT2,NODE3)
185   CONTINUE
```

Figure 6-6. Evaluate the neural network at a specific point.

6.4 TIFF Image of Shaped Neighborhood

The above techniques were used to render the shaped neighborhood shown in Figure 6-7. Normally, there is a section of code that limits the domain of the ray tracing by confining it to the region over which the network was trained. This is necessary since functional representation is undefined outside this region and there are no guarantees on results. In this particular image, the domain limiting code was turned off resulting in the artifact shown in the lower right hand corner. It is interesting to note how network training actually results in functional representation outside the training domain that seems to indicate some sort of extrapolation. Of course the representation is useless in these regions and is only included here as a curiosity.

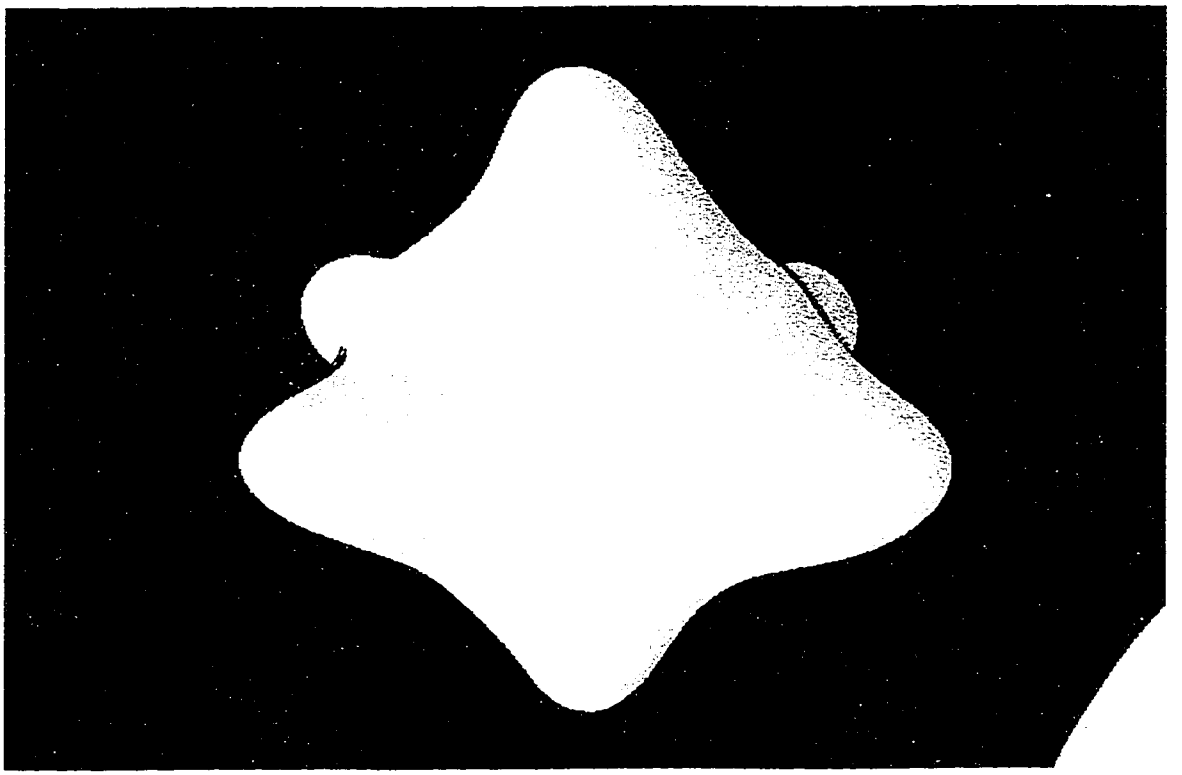


Figure 6-7. Shaped Neighborhood also showing artifacts.

Chapter 7
An Accelerated Convergence Algorithm

7.1 Introduction

Kohonen self-organizing feature maps are used to define a topology preserving mapping between a high dimensional neural network input space and a two dimensional output map space. Thus a representative set of points spanning the entire input domain is obtained, and various neural network training parameters such as training error and node gradient can be associated with their corresponding Kohonen map node. The map nodes are color-coded with respect to these associated parameters and presented visually as a 2D image, providing a means of quantifying the instantaneous network training activity. In this mode, the map provides a method of visualizing neural network convergence as represented by diminishing map activity. Further, the Kohonen map units represent vector prototypes over the input domain and the values associated with each map unit indicate training progress for a generalized region in the input domain.

This dissertation proposes a new method of network training which exploits the knowledge of current local and global error with respect to the entire input domain as provided by the SOFM map. In particular, a algorithm based on an alternating evaluation phase and training phase is implemented where the results of an evaluation phase feed the update equation in the following training phase. This method is compared to the popular Quick-Prop method using 2 different training examples. The SOFM based method defined in this implementation, while taking more machine cycles per training epoch, converges in substantially fewer epochs, resulting in an overall improvement for the examples given.

7.2 Neural Networks and Function Representation

Neural Networks are employed in a variety of applications including pattern recognition, classification, and other learning, generalizing, and predicting environments. This dissertation is motivated by neural networks as they pertain to function representation in multi-dimensional spaces. In particular, one can imagine an N-dimensional space with an M-dimensional vector associated with every point in that space. Note that M and N do not necessarily have to be equal, but it is reasonable that they could be as in a classical electric field in 3 space for instance. In such an environment, the neural network is an ideal vehicle to functionally represent the field, where the number of network input nodes equals the dimensionality of the space, and the number of network output nodes equals the dimensionality of the associated vector.

Thus the trained neural network becomes a black box capable of reproducing the vector field over the entire domain. The concept of a "shaped neighborhood" derives from this representation, and is in fact defined as a region in the space bounded by some particular vector field evaluation over the domain. For example, imagine a vector field defined on a 3-dimensional space where every point in the space is associated with a vector of magnitude equal to the distance to some reference point (called the origin) and whose direction is exactly away from the origin. A shaped neighborhood defined by all points whose associated vector has magnitude of 1.0 say, would be a sphere of radius 1.0 centered at the origin.

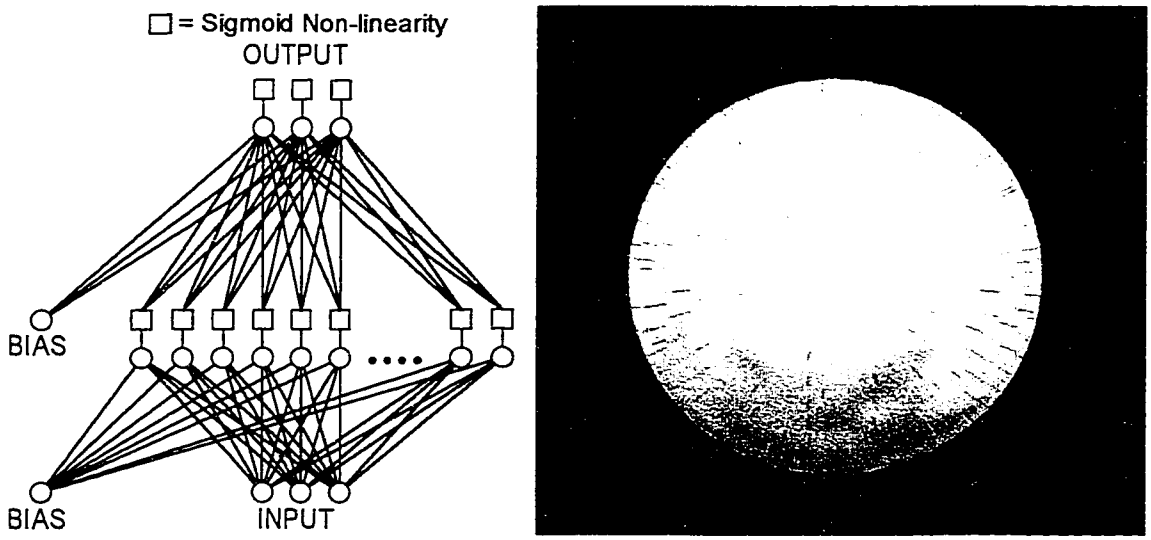
This dissertation is concerned with just such a neural network, its trainability, and its functionality. In particular, a one hidden layer perceptron is used to facilitate the implementation defined above. An external network visualization tool based on Kohonen self-organizing feature maps (SOFM) is presented, and an algorithm based on error information obtained from this map is used to accelerate convergence . Finally, training

results are compared to classical methods, and an example of a shaped neighborhood resulting from a network trained by the accelerated technique defined in this dissertation is shown and analyzed.

7.3 Layered Perceptrons and Back Propagation

The classical Multi-Layered Perceptron (MLP) is capable of "learning" generalized relationships between instances in various spaces. In set theoretic language, the trained network provides a mapping between a domain defined in some space to some range in a potentially different space. This mapping is learned through the training process where known relationships are presented to a network's input generating a error signal equal to the difference between actual and desired output, which in turn is back propagated through the network using a weight-adjusting algorithm. One of the most important features of a trained network is its ability to generalize over the input domain. In other words, the network will provide a reasonable mapping to input it has never experienced before.

This paper uses a layered perceptron consisting of three input nodes connected to a single hidden layer of twenty nodes, connected to a three-node output layer as shown in Figure 7-1(a). This network topology is specifically designed to learn relationships between points in 3D space and a given 3D vector associated with each point, thereby defining a vector field. This environment is ideally suited for defining "shaped neighborhoods" in these spaces as regions bounded by the locus of all points whose associated vector magnitude is less than some value α as in Figure 7-1(b). The associated vector also gives the boundary surface normal at any point which is useful for visualization.



(a) (b)

Figure 7-1 (a) NN architecture used to define "shaped neighborhoods" (b) A spherical shaped neighborhood defined by || vector || < α.

The learning process consists of tens of thousands of presentations of the training data delivered to a randomly initialized network, and the associated back propagation of the resulting error through the network in a weight-modifying algorithm. In practice, this is a relatively CPU intensive process, requiring fine tuning of the training parameters such as the learning rate parameter (LRP) and momentum. Significant work has already been accomplished by R. A. Jacobs [1988] with respect to understanding the reasons for the slow convergence in back propagation algorithms and also in developing training heuristics for accelerating the convergence. Specifically, techniques have been developed where each weight is assigned an individual LRP which changes over time in response to training progress [Hong et al. 1996] Other methods of modifying the LRP are based on the sign of the error gradient over several consecutive time steps.

7.4 Convergence Criteria

A number of methods exist for testing convergence to determining a training stopping point. As mentioned above, the activity produced by the weight update procedure is an indirect method of deducing convergence. When successive iterations of the training algorithm produce no appreciable change in the any of the network weights, it can be concluded that some type of minimum has been achieved. Another method would be to develop error metrics on the training data set directly as reproduced by the network. This paper proposes a visualization technique based on Kohonen Self-Organizing Feature Maps (SOFM) as a means of monitoring convergence, and indeed enhancing training through feedback to the weight update expression in the back propagation algorithm.

7.5 Feature Map Based Training Algorithm

Fuzzy logic methods for training parameter adaptation in neural network back propagation learning algorithms have been developed [Arabshahi et al. 1996]. The methods include developing fuzzy membership functions for the error signal and its derivatives, applying a set of heuristics defined in terms of If-Then rules, and deriving a crisp value for $\Delta\eta$ (change in LRP) which is then applied to the weight update equation in the BP algorithm. Other training methods such as Quick-Prop are based on the notion that the error surface is locally quadratic and attempt to jump to the minimum in one step.

Kohonen Self-Organizing Feature Maps transform an input of arbitrary dimension into a one or two-dimensional discrete map subject to a neighborhood preserving, topological constraint [Kohonen 1984]. The unsupervised, self-organizing competitive algorithm generates a mapping of an input signal vector in some high-dimensional space onto a one or two-dimensional topological space which is ideally suited for visualization. SOFM and

MLP have been used in hybrid networks before, typically in a cascade architecture, to accomplish various tasks [Lee et al. 1996]. This dissertation defines a hybrid architecture using a SOFM data representation external to the NN to define a "next step size" in the evaluation phase which feeds the weight update algorithm in the train phase.

This dissertation proposes a new training method using a Kohonen Self-Organizing Feature Map (SOFM) to create a generalized characterization of the NN training state over the entire input domain as a global view. This view provides an opportunity to expand the context of the error associated with regions of input and accelerate the NN convergence through the SOFM based, weight update algorithm. In particular, the algorithm is based on an evaluate, then train phase for each epoch cycle. During the evaluation phase, the error for each SOFM map node is determined by passing the SOFM map nodes (input data prototypes) through the NN in a single, feed forward pass. In the train phase, this prototype error defines the weight update factor used by the back propagation algorithm.

The Kohonen map

A Kohonen SOFM is created over the neural network training dataset thus producing a representative set of 2-dimensional, topology preserving tokens spanning the input training data. Figure 7-2 shows a SOFM "unfolding" as it generalizes over points randomly distributed throughout a domain defined in 2-space. This dissertation exploits this map by associating various types of training information with these representative map nodes. In particular, the error with respect to the local minimum for both the current cycle and the previous cycle are stored.

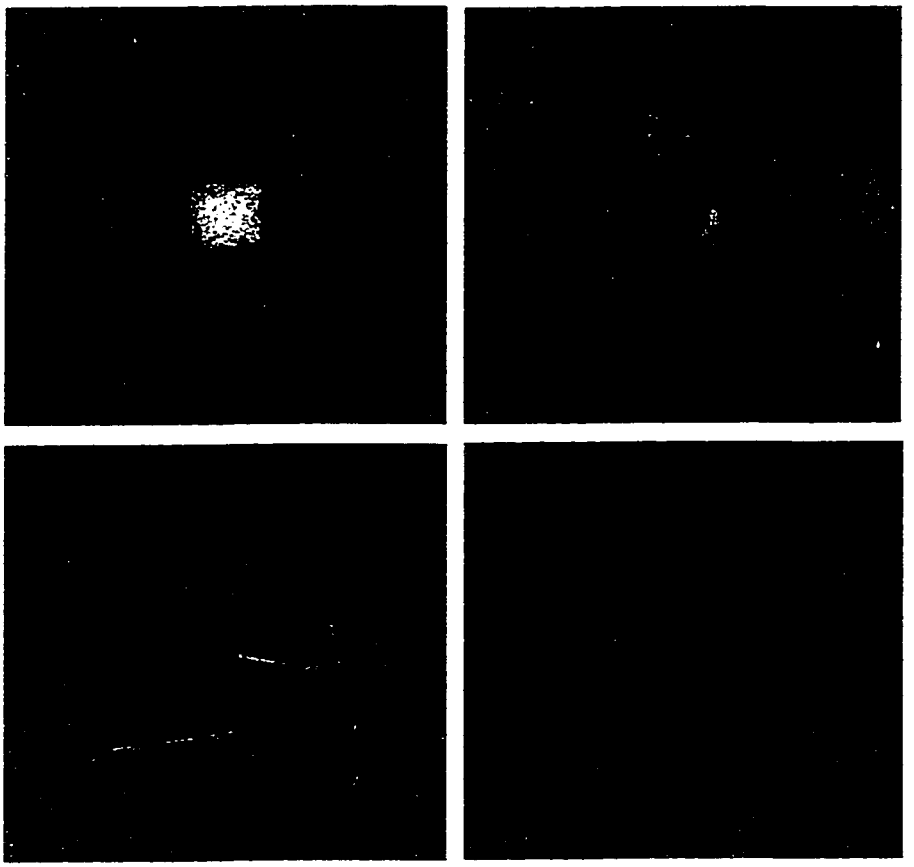


Figure 7-2. A Kohonen SOFM "unfolding" to generalize random points distributed uniformly throughout a 10x10 domain of interest. From left to right, top to bottom: initialization, 10 epochs, 20 epochs, and convergence at 500 epochs.

The Kohonen map is visualized graphically where areas of high training activity are readily apparent such as shown in Figure 7-3. These areas of activity are characterized by high node gradients and high error color coded red in Figure 7.3. Thus, at any point during training, a visual map can be obtained characterizing the quality of the training with respect to the entire input domain.

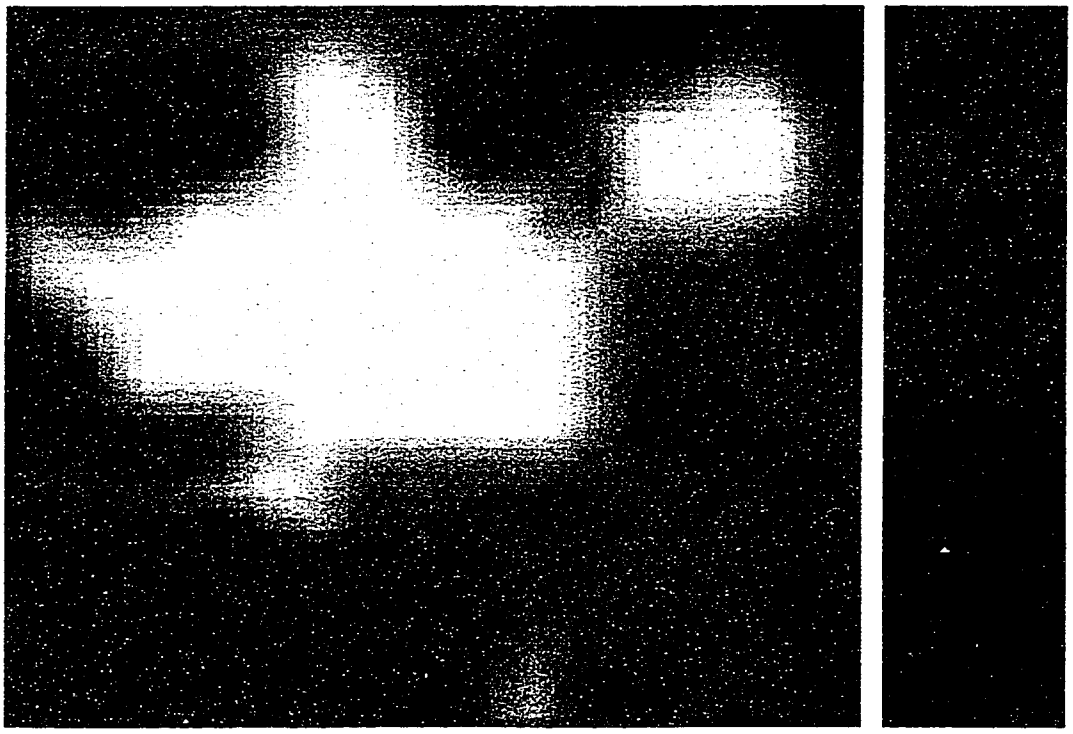


Figure 7-3. Example of SOFM based image used to visualize NN training progress with respect to the input domain. Map nodes span the domain and are color-coded to represent training activity.

The SOFM based Weight Update Algorithm

The SOFM map establishes a global view of error during the evaluation phase of each training epoch. At this time, all SOFM prototypes are passed through the NN being trained, and the difference between the error associated with each prototype and the local minimum error is determined. This error, as well as the error for the previous training epoch are stored with the prototype. Finally, during the training phase of the cycle, weights are updated in classical back propagation fashion according to the expression given in equation (9). This equation essentially develops the ratio of the next step size to

the current step size based on how effective the current step was in moving towards the error associated with the local minimum. This ratio becomes the scaling factor applied to the current change in weights:

$$\Delta W(\text{current}) = \frac{\text{current error}}{\text{previous error - current error}} * \Delta W(\text{previous}) \qquad (9)$$

This ratio is further visualized in Figure 7.4. Here, the weight update from the previous cycle resulted in a corresponding step toward the local minimum as indicated by red. The current step would ideally continue the rest of the way to the local minimum error as indicated by blue. Thus the ratio defined by blue over red should be applied to the previous weight update to achieve the current weight update.

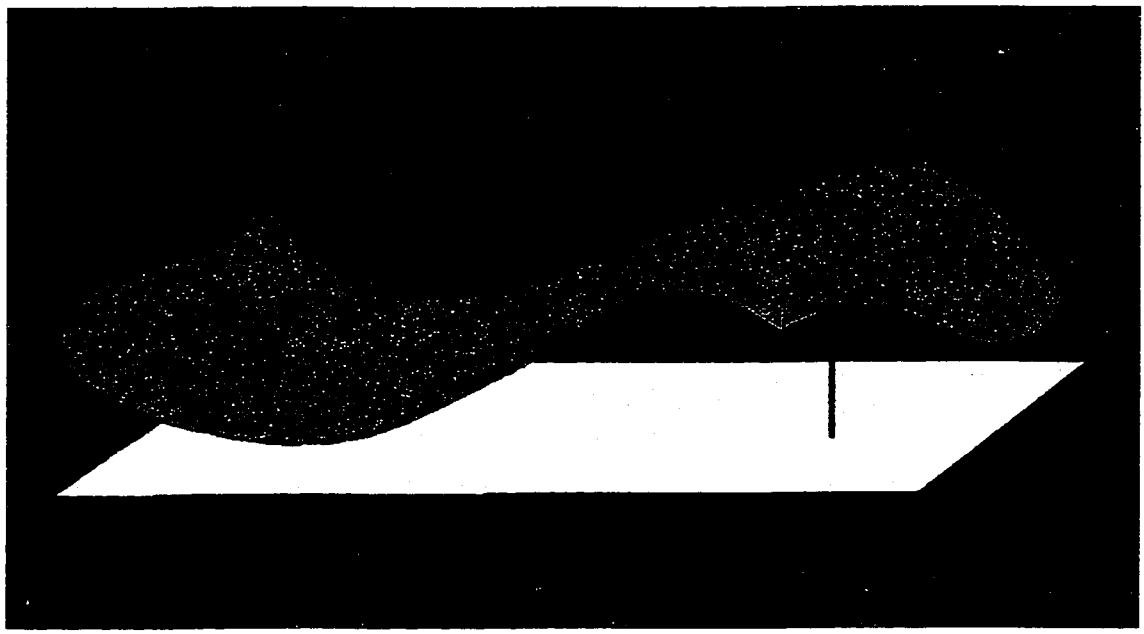


Figure 7-4. SOFM based weight update

The algorithm of alternating phases of evaluate, then train, is implemented using the architecture indicated by Figure 7-5. Here, the training data is used to produce a SOFM map of training prototypes. These prototypes are passed through the NN during the evaluation phase establishing a minimum error over the entire input domain. During the training phase, individual training vectors reference this minimum error, and the ratio of current error to previous error with respect to this minimum error defines a weight update value for that particular training vector.

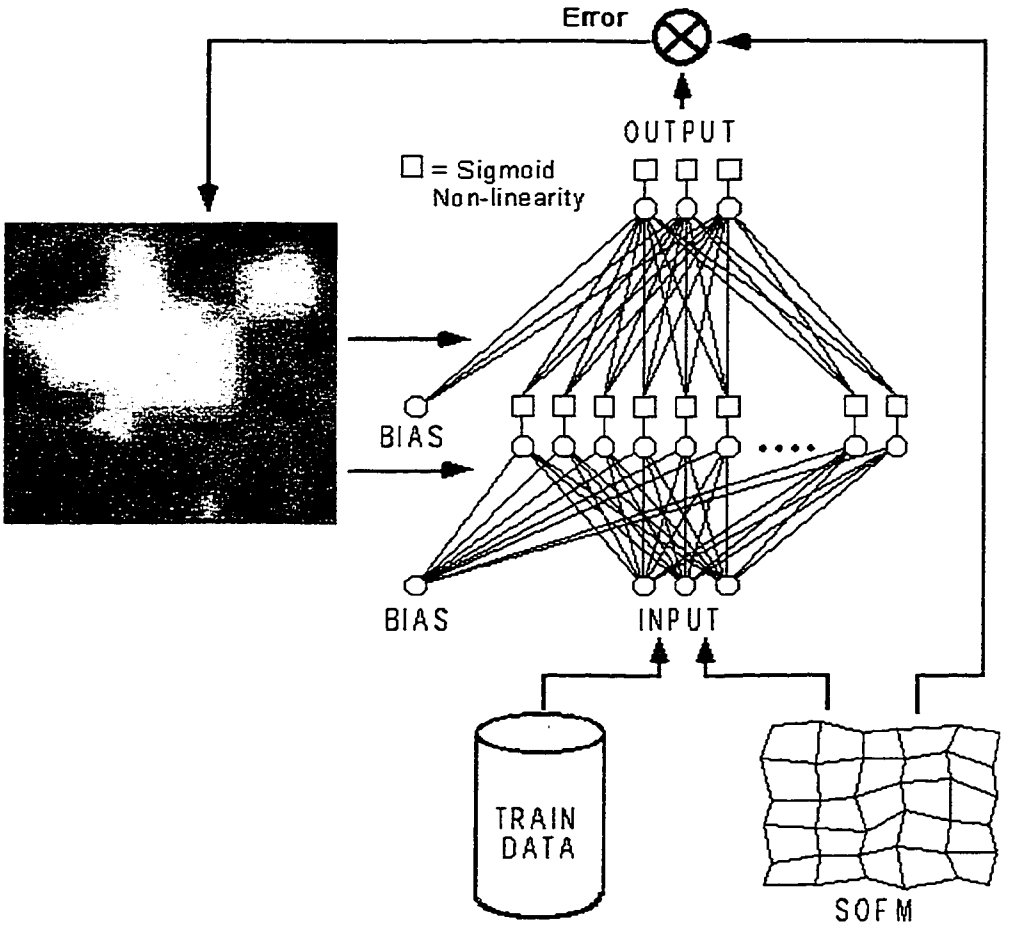


Figure 7-5. Schematic of SOFM based convergence acceleration

7.6 Quick-Prop Comparison

The SOFM based training method defined above was compared to the Quick-Prop method widely used for NN training. The Quick-Prop method assumes a quadratic error surface for the error with respect to weights. Using this assumption, a next step size is calculated based on previous and current error gradients which attempts to jump to the local error minimum in a single step. This is diagrammed in Figure 7-6. Again, the ratio of blue over red is applied to the previous weight update to achieve the current weight update factor according to equation (10) where s(t) and s(t-1) are current and previous error gradients.

$$\Delta W(t) = \frac{s(t)}{s(t-1) - s(t)} * \Delta W(t-1) \qquad (10)$$

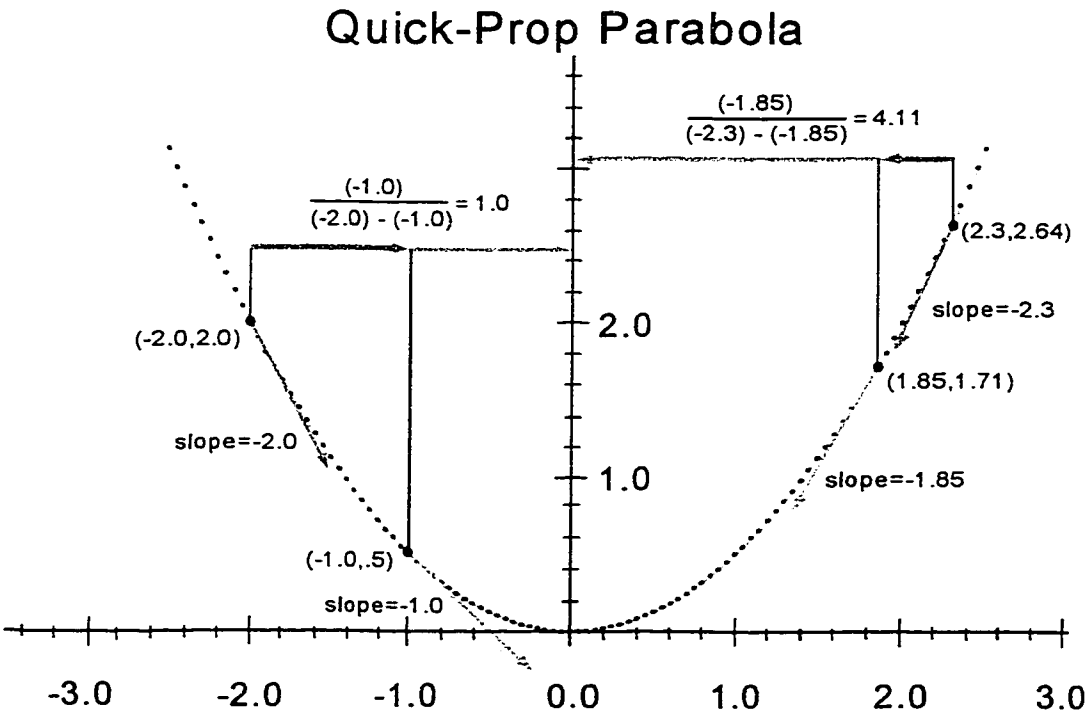


Figure 7-6. Parabolic based step size used by Quick-Prop

7.7 Feature Map Visualization

This paper presents an alternative method to determining training stopping criteria. The method derives directly from the SOFM characterization of the prototypical input domain. Figure 7-7 shows a series of snapshots of the SOFM map taken during a complete network training. The first frame shows the randomness associated with the network initialization. This resolves relatively quickly to show areas of high activity. As the training proceeds, the level of activity subsides and a uniform map of a minimum intensity evolves indicating convergence.

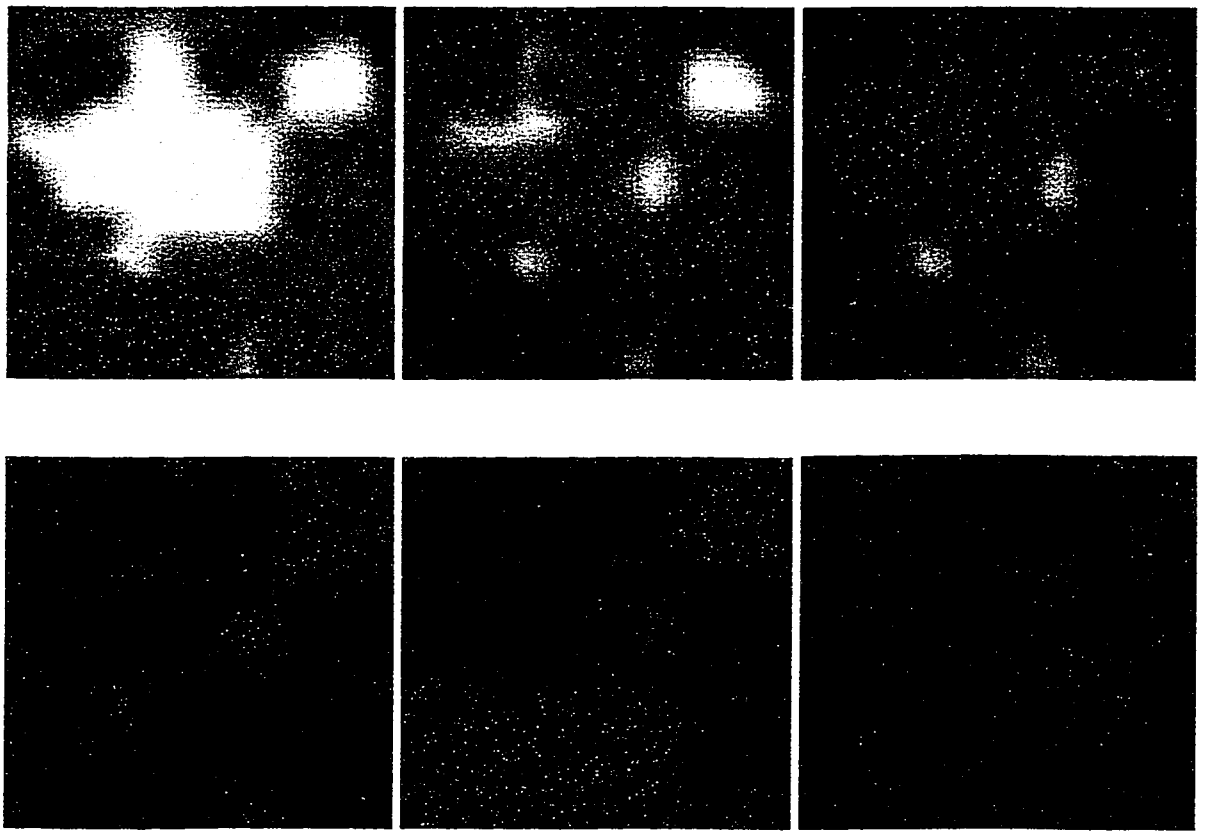


Figure 7-7. SOFM map progressing through various phases of NN convergence. From left to right, top to bottom: initialization, 5 epochs, 20 epochs, 500 epochs, 5000 epochs, and convergence at 12,200 epochs.

The time scale for the progression shown in Figure 7-7 is weighted towards early training epochs where much of the large scale activity. This is qualitatively correlated with the early activity seen in a weight trace such as shown in Figure 7-8.

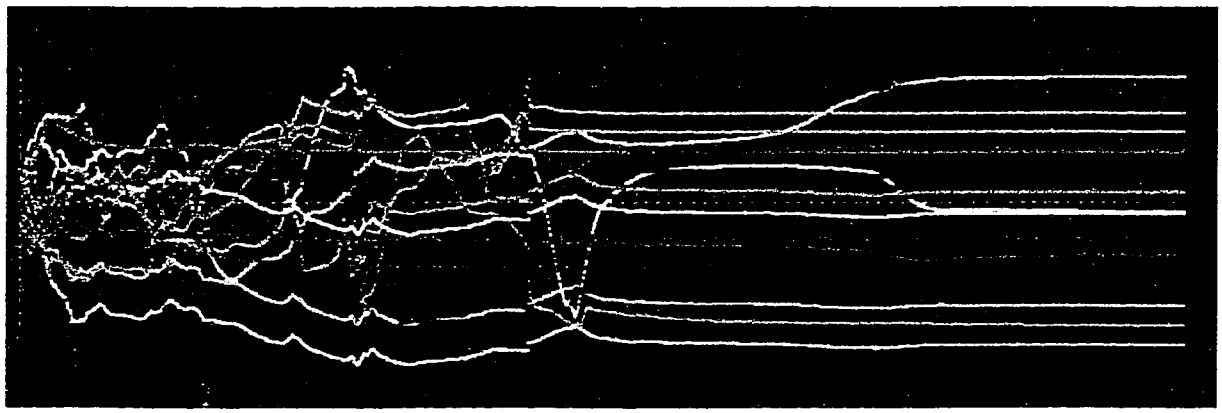


Figure 7-8. Graphical Visualization of Weight Evolution.

7.8 Accelerated Convergence

Neural Networks are trained over some input domain. The Kohonen SOFM provides a means of characterizing this entire domain as a set of input prototypes. These prototypes can be passed through the NN being trained at regular intervals to determine the progress of training. This progress can be easily visualized as a two-dimensional map since the SOFM map is in fact a 2D representation of the higher dimension space. This visualization provides the opportunity to assess convergence, as well as defining an input domain spanning error signal which can be exploited by the training weight update algorithm to accelerate training convergence.

7.9 Results

Verification of Methods: Case 1. Shaped Neighborhoods

The SOFM based training methods presented in this dissertation were verified on the neural network topology define in the introduction. The training data for these trials are based on a set of 300 discrete points and associated vectors characterizing a "shaped neighborhood" in 3 dimensions. This shaped neighborhood is shown in Figure 7-9. It is best thought of as defining a "region of interest" in 3D space.

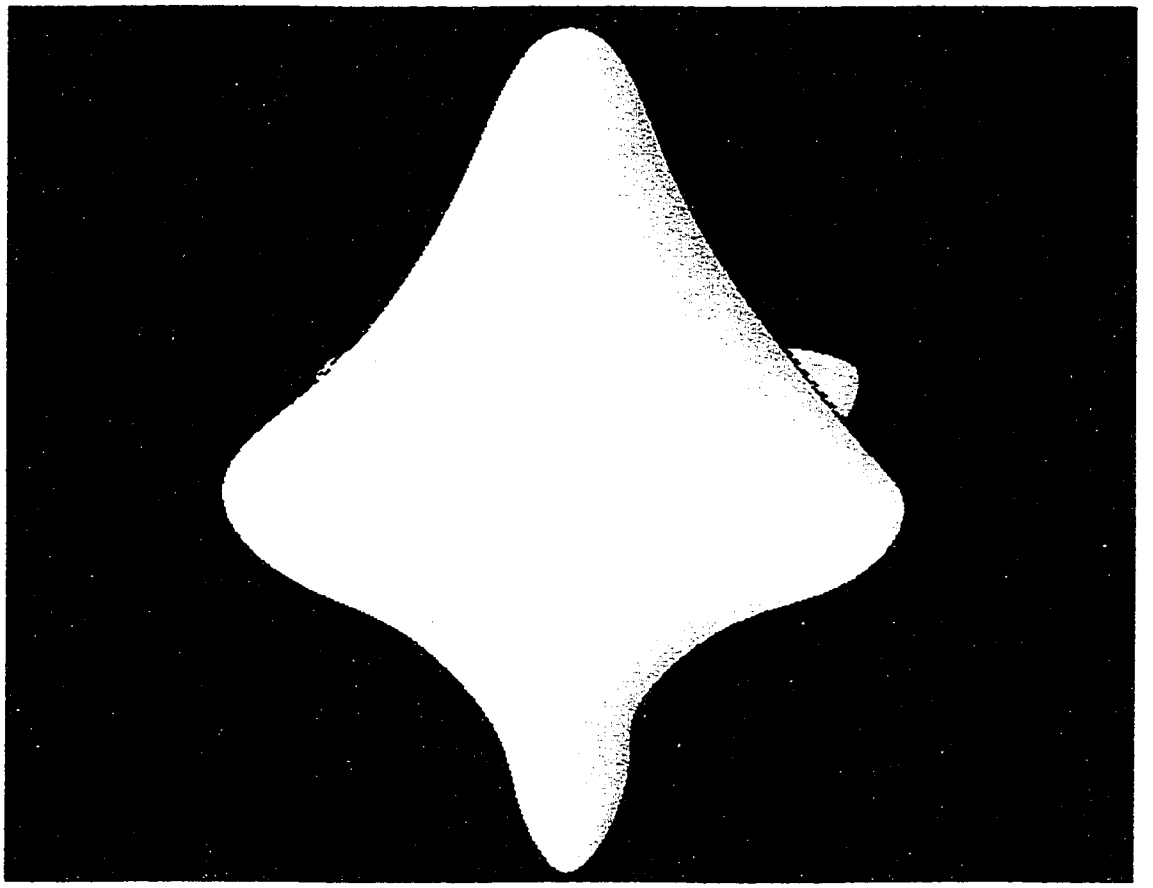
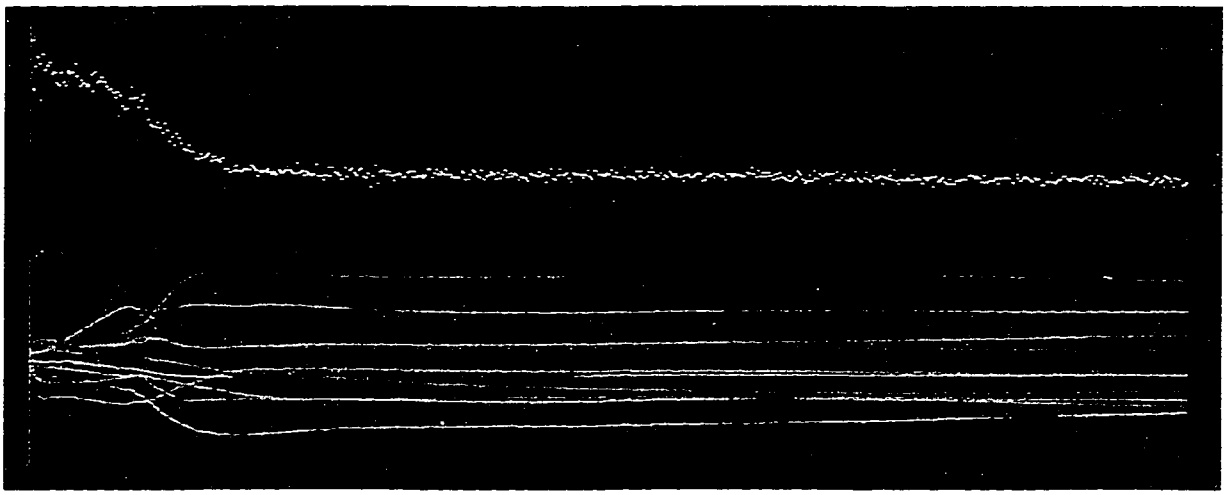
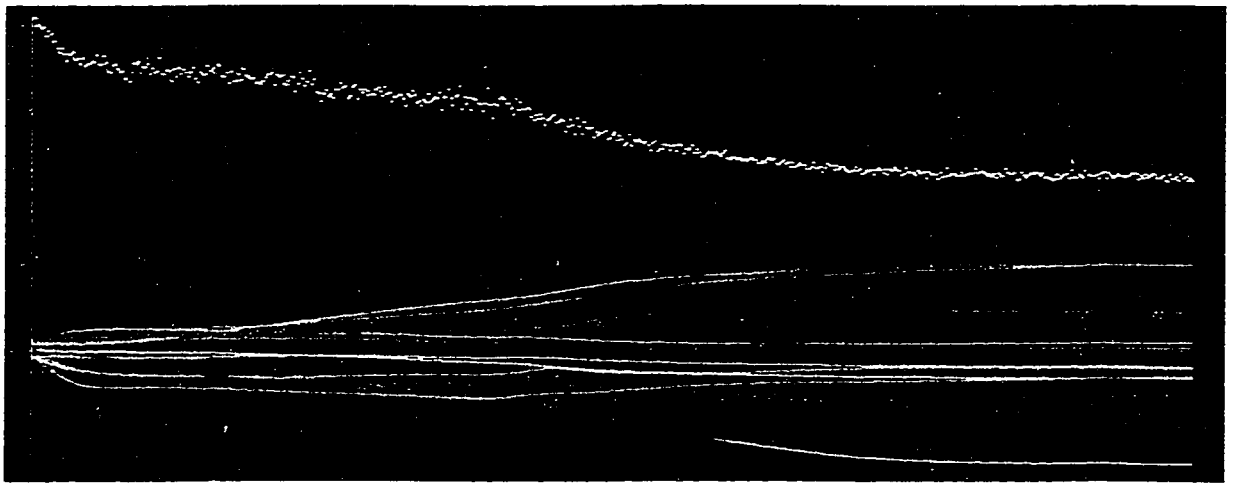


Figure 7-9. A Shaped Neighborhood in 3 Dimensions.

This "shaped neighbor" was visualized through a ray tracing algorithm coupled to the trained neural network. The NN output is in fact the surface normal which is used in a simplified Phong shading technique. Figure 7-10 shows the weight evolution visualizations for two trials, (a) with SOFM based weight update, and (b) with back propagation training algorithm.



(a)



(b)

Figure 7-10. NN Convergence Comparison: (a) with SOFM based update, (b) standard back propagation. Top curve in each case is total error. Bottom curves are network weights.

Interpretation of Results

The curves in Figure 7-10 show a comparison between two neural networks trained over identical input data sets. The basic NN architectures were also identical. The network in (a) was trained using the SOFM based update methods defined in this dissertation, while the NN in (b) used classical back propagation. The total average error for the networks during training is shown in the top half of each diagram while the bottom half shows the evolution of the weight parameters for each network. Total average error = 0.002 (arbitrary units) was chosen as a standard of convergence. The SOFM based network achieved this standard in 12,200 epochs while the standard back propagation network required 47,500 epochs to achieve the results.

Notice both training periods were extended to 50,000 epochs. This was done to provide a means of comparing CPU metrics. Recall that SOFM algorithm involves "evaluating" the complete set of input prototype vectors at the beginning of each epoch of training (single feed forward pass only). This of course necessitates machine cycles not otherwise required by the un-sensitizing algorithm. The following metrics were obtained on an IBM 3090 MVS mainframe system:

1) CPU required for 50K epochs using SOFM: 5 min. 15.96 sec.
2) CPU required for 50K epochs using standard BP: 3 min. 9.71 sec

It can be seen that the SOFM algorithm requires 66% more machine cycles per training epoch. Of course the pay back derives from the fact that less epochs are required. Thus, an improvement of 234% is achieved by SOFM based algorithm:

1) SOFM: 12.2K epochs X 0.00010532 min/epoch = 1.285 minutes
2) BP: 47.5K epochs X 0.00006324 min/epoch = 3.003 minutes

One finally interesting "discovery" was made during the verification trials. The SOFM visualization shown in Figure 7-7 is the visual series generated by the trial with feedback as defined above for the shaped neighborhood shown in figure 7-9. Recall that this SOFM map is in fact a topology preserving representation of input space. The areas of high activity in the early stages of training seem to roughly resemble a projection of the shape being learned. This was qualitatively verified with a number of other shaped neighborhoods.

Verification of Methods: Case 2. Iris Classification

The SOFM based methods presented in this dissertation were also verified using the classification of iris data [R. A. Fisher, 1950]. This data contains 3 classes of 50 instances each, where each class refers to a type of iris plant. In this case, it is a given that one class is linearly separable from the other 2, while the latter 2 are not linearly separable from each other. A portion of the iris data is shown in Table 3.

Table 3. Iris Classification Data

sepal len.	sepal width	petal len.	petal width	classification
5.1	3.5	1.4	0.2	Iris Setosa
4.9	3.0	1.4	0.2	Iris Setosa
5.7	3.0	4.2	1.2	Iris Versicolour
5.7	2.9	4.2	1.3	Iris Versicolour
5.7	2.5	5.0	2.0	Iris Virginica

The system was arranged as shown in Figure 7-5 with the external SOFM map monitoring the input domain while providing a step size information to the weight update expression.

Figure 7-11 shows a comparison for two curves for three different methods of training. The red traces indicate RMS error as well as discrete classification error for a NN using classical back propagation. The green traces correspond to an equivalent NN using Quick-Prop. The blue traces are achieved using the SOFM based training methods presented in this dissertation.

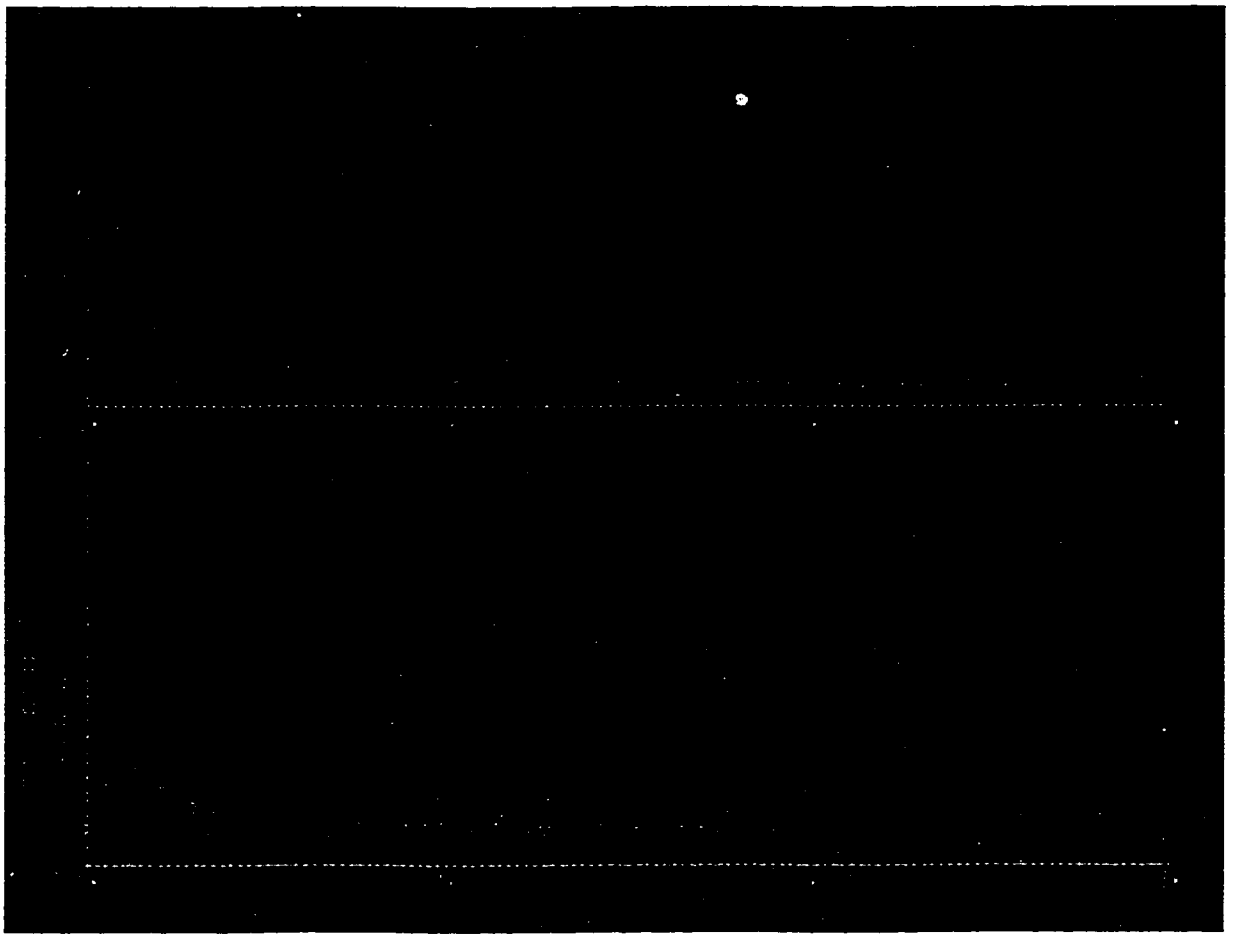


Figure 7-11. Convergence comparison for SOFM, Quick-Prop, and classical back propagation methods.

Interpretation of Results

As can be seen from the graphs, the SOFM based method reaches a final classification in approximately 1540 epochs while the exact same network using Quick-Prop takes 1990 epochs. Note that the discrete error for both systems converges on 2 miss-classified instances. This is a well know result for this particular data set owing to the lack of linear separation in the training data.

As a final comparison, the CPU time for each training method is presented below. This runs were performed on an IBM 3090 mainframe system under the MVS operating system. Note that while the SOFM based method presented in this dissertation takes slightly more cycles per iteration than Quick-Prop (1.2%). It does however, converges in less epochs for a total improvement of 21.6 % over Quick-Prop. The slightly longer time per epoch is due to the fact that the SOFM prototypes are evaluated once each epoch to provide network error information used by the update algorithm.

```
Quick Prop

SYS001I *****   MVS 390  LEVEL: 9820 OS/390R5 NODE: BGDX
SYS001I *****   DATE:05/16/99  *****  BOEING
SYS010I  JOB/JOB54777/GWA006R   TOTAL-CRU( 14.41)
ID=BGD2  DATE=(99.139 -  5/19/1999)
IEF376I  JOB/GWA006R /START 1999139.1921
IEF376I  JOB/GWA006R /STOP  1999139.1923
CPU: 0MIN 54.63SEC     SRB: 0MIN 00.11SEC
READY

SINGE-LAYER PERCEPTRON (Quick Prop)

EPOCH=       1  ERRCNT=     125
EPOCH=       5  ERRCNT=      84
EPOCH=      10  ERRCNT=      75
EPOCH=      15  ERRCNT=      54
EPOCH=      20  ERRCNT=      49
EPOCH=      25  ERRCNT=      48
...
...
EPOCH=     185  ERRCNT=      10
EPOCH=     190  ERRCNT=      10
EPOCH=     195  ERRCNT=      10
...
...
EPOCH=    1975  ERRCNT=       4
EPOCH=    1980  ERRCNT=       4
EPOCH=    1985  ERRCNT=       4
EPOCH=    1990  ERRCNT=       3
EPOCH=    1995  ERRCNT=       3
...
...
EPOCH=    2990  ERRCNT=       3
EPOCH=    2995  ERRCNT=       3
EPOCH=    3000  ERRCNT=       3

*************************************************************
*       NO ERROR DETECTED
*************************************************************
```

SOFM Based Feedback

```
SYS001I *****   MVS 390  LEVEL: 9820 OS/390R5 NODE: BGDX
SYS001I *****   DATE:05/16/99  *****  BOEING
SYS010I  JOB/JOB54802/GWA006R   TOTAL-CRU( 14.60)
ID=BGD2  DATE=(99.139 -  5/19/1999)  V(95)
IEF375I  JOB/GWA006R /START 1999139.1947
IEF376I  JOB/GWA006R /STOP  1999139.1948
CPU: 0MIN 35.34SEC     SRB: 0MIN 00.13SEC
READY

SINGE-LAYER PERCEPTRON (SOFM BASED FEEDBACK)

EPOCH=        1  ERRCNT=   100
EPOCH=        5  ERRCNT=    55
EPOCH=       10  ERRCNT=    50
EPOCH=       15  ERRCNT=    50
EPOCH=       20  ERRCNT=    50
EPOCH=       25  ERRCNT=    50
...
...
EPOCH=      150  ERRCNT=    10
EPOCH=      155  ERRCNT=    10
EPOCH=      160  ERRCNT=    10
...
...
EPOCH=     1510  ERRCNT=     4
EPOCH=     1520  ERRCNT=     4
EPOCH=     1530  ERRCNT=     4
EPOCH=     1540  ERRCNT=     3
EPOCH=     1550  ERRCNT=     3
EPOCH=     1560  ERRCNT=     3
...
...
EPOCH=     2990  ERRCNT=     3
EPOCH=     2995  ERRCNT=     3
EPOCH=     3000  ERRCNT=     3


**************************************************************
*       NO ERROR DETECTED
**************************************************************
```

Back Propagation

```
SYS001I  *****   MVS 390  LEVEL: 9820 OS/390R5 NODE: BGDX
SYS001i  *****   DATE:05/16/99   ***** BOEING
SYS010I  JOB/JOB55002/GWA006R   TOTAL-CRU(  8.96)
ID=BGD2  DATE=(99.139 -  5/19/1999)  V(95)
IEF375I  JOB/GWA006R /START 1999139.2108
IEF376I  JOB/GWA006R /STOP  1999139.2109
CPU: 0MIN 53.95SEC     SRB: 0MIN 00.12SEC
READY

SINGE-LAIER PERCEPTRON (Back Propagation)

EPOCH=           1   ERRCNT=         250
EPOCH=           5   ERRCNT=         155
EPOCH=          10   ERRCNT=         150
EPOCH=          15   ERRCNT=         150
EPOCH=          20   ERRCNT=         150
EPOCH=          25   ERRCNT=         150
...
...
EPOCH=         520   ERRCNT=          10
EPOCH=         525   ERRCNT=          10
EPOCH=         530   ERRCNT=          10
...
EPOCH=        1825   ERRCNT=           6
EPOCH=        1830   ERRCNT=           6
EPOCH=        1835   ERRCNT=           6
EPOCH=        1840   ERRCNT=           6
EPOCH=        1845   ERRCNT=           6
EPOCH=        1850   ERRCNT=           6
...
...
EPOCH=        2990   ERRCNT=           6
EPOCH=        2995   ERRCNT=           6
EPOCH=        3000   ERRCNT=           6


**********************************************************
*       NO ERROR DETECTED
**********************************************************
```

Chapter 8
A Detail Part Design Retrieval System

8.1 Introduction

The methods discussed in chapters above were used to develop the detail part design retrieval prototype described in this chapter. This system features an IBM DB2 relation database running in a Sun Solaris environment. Engineering detail part representation is supported through the notion of an 8-dimensional hypercube where each axis corresponds to a particular part parameter such as height, length, volume, etc.. This database is driven off the configuration management system responsible for releasing engineering datasets. An Engineer involved in preliminary design generates a representation (set of parameters) for a desired detail part that meets a certain set of design requirements. This part parameterization is used to access the database through SQL (Standard Query Language). Results from the SQL query are points representing detail part instances, which lay in a subspace of the hypercube. The subspace is characterized as rectilinear due to the nature of SQL and represents a first cut at discovering existing detail parts which meet the design criteria.

This subspace is subsequently processed by a series of stackable filters including analytic, fuzzy logic, and neural network based shaped neighborhoods, which are in turn delivered a refined set of similar part candidates back to the design engineer. Figure 8-1 shows the basic layout for this system. Ultimately, through the use of various filters and combinations of filters, a reasonable set of suitable detail part candidates is delivered to the user. These candidates can then be visualized and a final determination made with respect to their suitability.

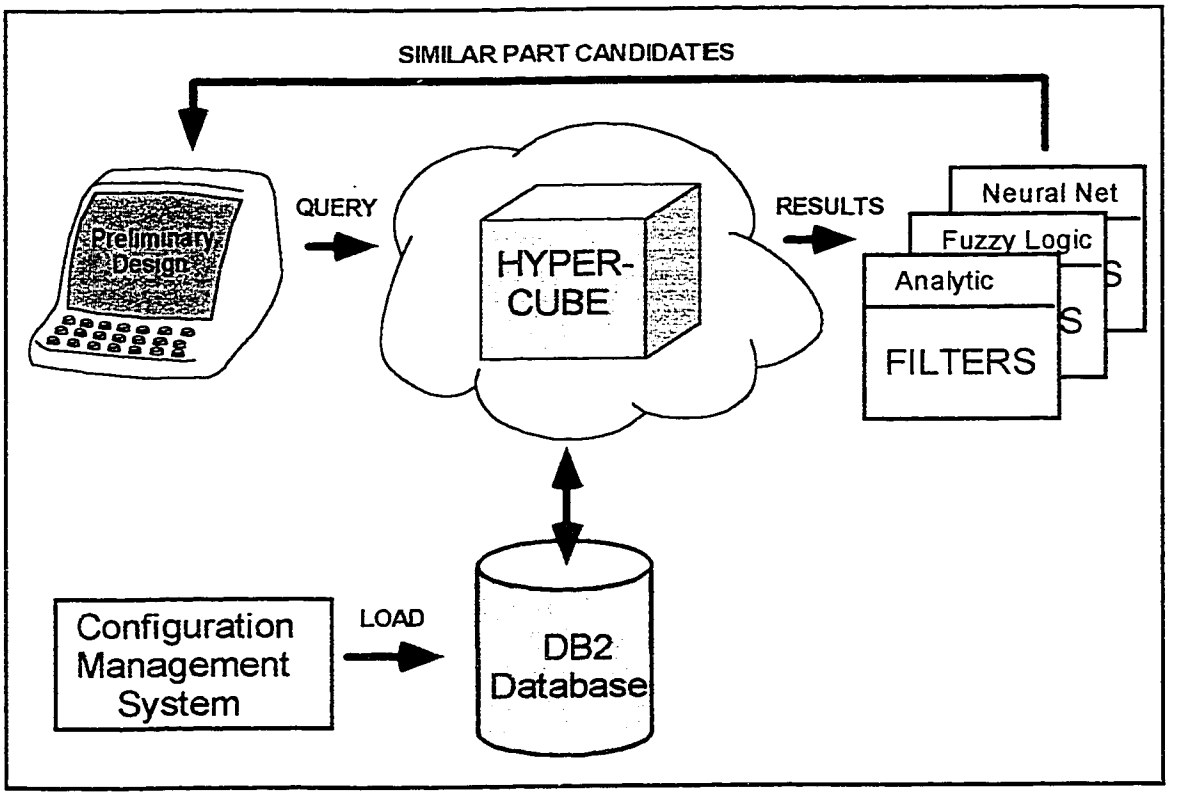


Figure 8-1. Overview of Design Retrieval System.

8.2 System Architecture

The design retrieval system prototyped in this dissertation is centered around the IBM DB2 Version 2.1 relational database hosted on a Sun Enterprise 3000 workstation. As with any relational database, data is organized around tables. The main objective for the prototype included similar part retrieval, and as such, database parameters included fields to support this end. Secondary objectives included visualization of "discovered" similar parts, as well as a metrics system for monitoring queries, matches, and other user information. This objectives are reflected in the table structure shown in Figure 8-2.

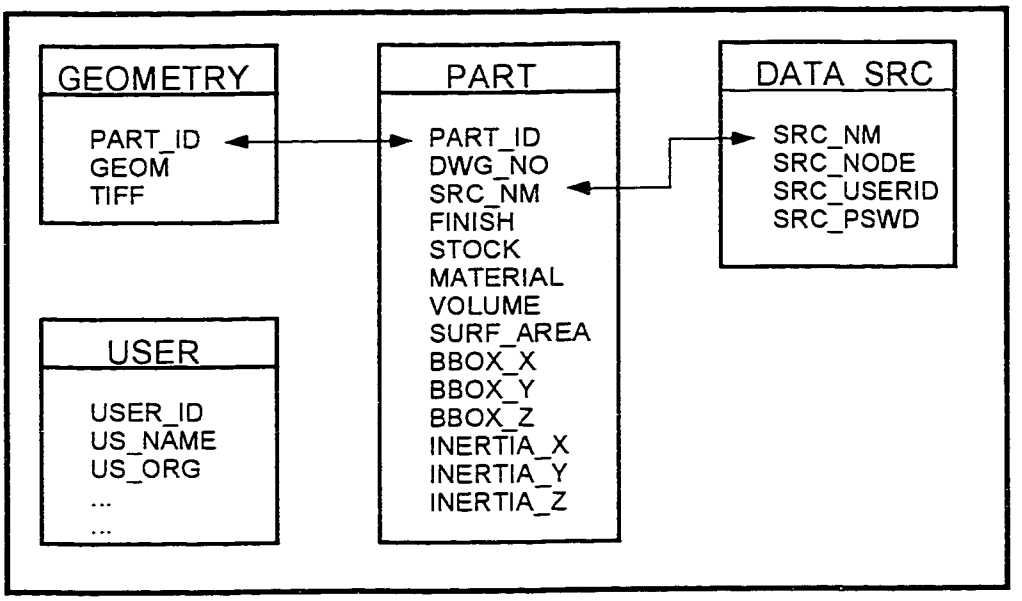


Figure 8-2. DB2 Relational Database Table Structure.

The main table named PART represents an instance of a detail part. The eight properties, one for each axis of the hypercube, are defined in this table along with pointers to the CAD (Computer Aided Design) definition of the part as it is defined in the Configuration Management System. The table named GEOMETRY contains information used to visualize the candidate parts identified as similar. Various user information supporting usage metrics is contained in the USER table.

Cascading Filters

A target part is identified by a design engineer according to some design criteria. Nominal values are calculated from the target and formulated into an SQL query against the database. The returned report is considered the superset of potential similar parts and is characterized by its classic rectilinear region in hyperspace. For typical ranges of parameter variability, hundreds of candidates may be returned. The results of this

preliminary filter are cascaded to a series of appropriate filters selected by the user to refine the search and ultimately narrow the results to 5 or 10 candidate parts. These parts are then visualized via graphic pointers to geometry and TIFF files stored in the database GEOMETRY table, and ultimately selected or rejected by the design engineer as possible substitute parts for their current design requirements.

Shaped Neighborhoods

The motivation for shaped neighborhoods as filters in high dimension space is simply the desire for a completely general definition of the relationship between parameters in this space. On one extreme, the parameter variations defined by the rectilinear regions derived from SQL filtering exhibit complete independence in that all combinations of all parameter ranges are valid. In high dimension space, this yields unsuitable results at the "corners" of the rectilinear regions. On the opposite extreme are the neural net based shaped neighborhoods defined in this dissertation, where every parameter is given as a function of every other parameter throughout their range. This inter-parameter functional relationship is captured by the neural net.

The primary utility derived from shaped neighborhoods is their ability to refine the acceptable range of variability of all parameters with respect to each other. Recall that the parameters in this application are detail part surface area, volume, length, width, height, and moments of inertia around the principal axises. Thus concepts like length proportional to strength (cross-section), area constrained by weight, and near-nominal verses far-nominal parameterizations are possible. In fact, shaped neighborhoods are motivated by a desire for complete generality in expressing inter-parameter relationships, and exploiting these relationships is support of a similar detail part discovery methodology.

8.3 The Physical Prototype

The prototype retrieval system was loaded using commercial airframe structural detail part data of the type described in section 1. C code was written to access the configuration management system's released engineering datasets and format the appropriate information as clear text which in turn was loaded into the DB2 table structure of Figure 8-2. Once loaded, a target part was identified and an SQL query was used to return a superset of candidate parts. This candidates were filtered through a neural network based shaped neighborhood designed to constrain overall parameter variability near-nominal values while allowing individual parameters to range to greater extremes.
The resulting candidates were visualized and compared to other members of the superset which were rejected by the shaped neighborhood filter.

Input data

Figure 8-3 shows a small section from the clear text load file generated by accessing the configuration management system. There are primarily two opportunities for detail part parameter capture in support of the design retrieval system defined here. Initially, a software agent traverses the configuration release system and processes each part to generate a "flat file" as shown in Figure 8-3. Once the initial load is complete, a second software agent monitors ongoing releases and generates updates to the database on a periodic basis. One feature of this system is the elimination of the need for extensive network training operations required by other clustering type approaches. Finally, a third class of software agents are active to maintain the integrity of the database. These agents are involved with retrieving missing information for incomplete records from other remote systems, and other database maintenance duties.

```
**********************************************************************
PART ID :   1627
TITLE   : 123X4567-001
TYPE    : SOLIDE
SURFACE AREA      :      14.60640426
VOLUME            :       0.43788669
MOMENT OF INERTIA :       0.68371844       0.64895725       0.10345271
BOUNDING BOX      :       0.89999998       0.75790137       2.06534863
**********************************************************************
PART ID :   1628
TITLE   : 777X4567-002
TYPE    : SOLIDM
SURFACE AREA      :     578.18550820
VOLUME            :       9.20885025
MOMENT OF INERTIA :     526.04929838     174.44478365     351.60608637
BOUNDING BOX      :       0.03200000       9.57613850      12.34759045
**********************************************************************
PART ID :   1629
TITLE   : 654X4567-003
TYPE    : SOLIDE
SURFACE AREA      :     578.20388647
VOLUME            :       9.20913345
MOMENT OF INERTIA :     526.08104429     174.45713078     351.62548520
BOUNDING BOX      :       0.03200000       9.57629967      11.76204872
**********************************************************************
PART ID :   1630
TITLE   : 444X5555-001
TYPE    : SOLIDM
SURFACE AREA      :    1084.87324449
VOLUME            :      33.77771886
MOMENT OF INERTIA :   18827.90721815   16916.38883864    1911.54072347
BOUNDING BOX      :       0.06299999      18.24826050      42.92649841
**********************************************************************
PART ID :   1631
TITLE   : 555C6678-033
TYPE    : SOLIDE
SURFACE AREA      :    1084.77899095
VOLUME            :      33.77465997
MOMENT OF INERTIA :   18827.29066582   16915.84962523    1911.46338254
BOUNDING BOX      :       0.06299999      18.24598694      43.28721619
**********************************************************************
```

Figure 8-3. Sample of data used to load the DB2 Relational Database.

Standard Query Language (SQL)

Standard Query Language (SQL) is used as a preliminary filter to identify the superset of candidate detail parts defined by some target part plus and minus some range on each of the property parameters. This superset is characterized as the rectilinear region in the hyper space centered around the target part location. The pertinent SQL syntax is shown in Figure 8-4.

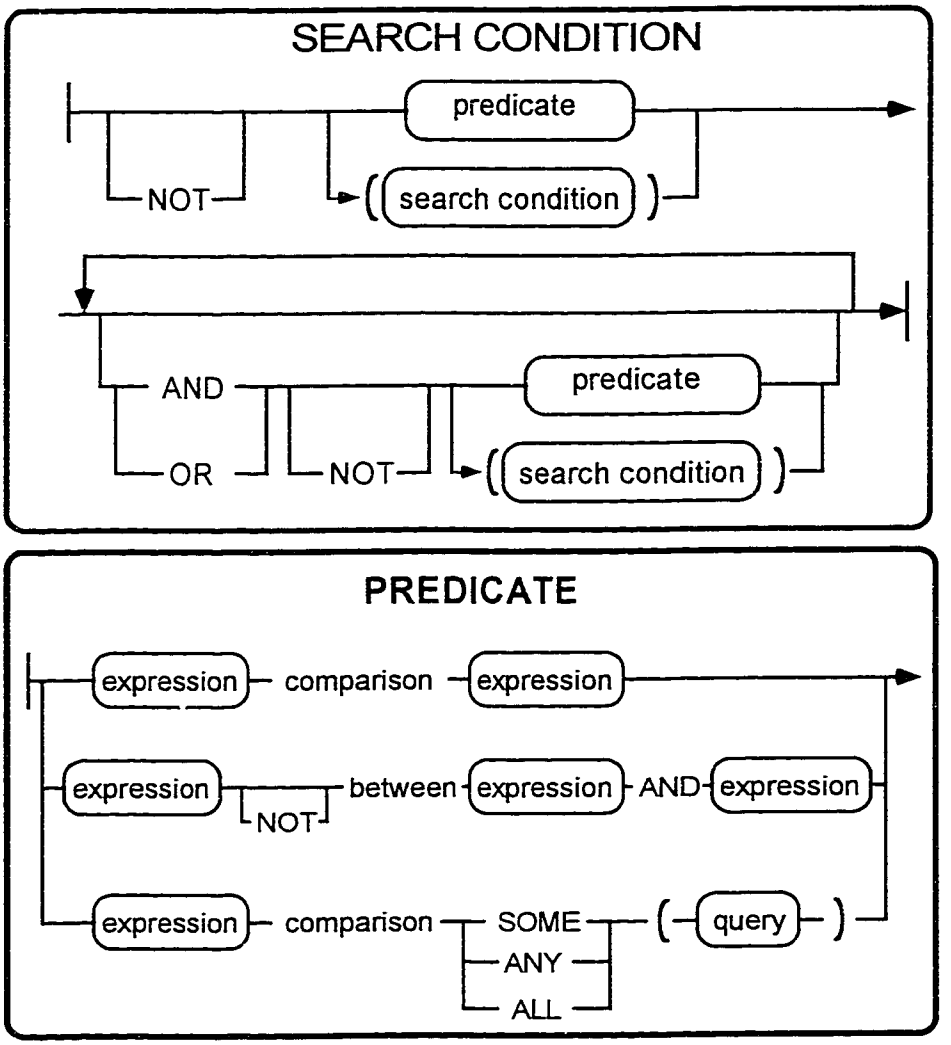


Figure 8-4. SQL syntax used for preliminary filtering.

A typical SQL query would select elements (detail parts) from the PART table according to some criteria. Commands can be submitted from the Command Line Processor (CLP), read from a file, or embedded in a programming language supporting higher level structure such as C. A typical command might take the form:

```
SELECT "PART_ID" FROM PART WHERE "SURF_AREA"
BETWEEN 10.0 AND 20.0;
```

The result set returned by such a command is passed to subsequent analytic, fuzzy logic, and neural net filters which further refine the results set.

The C Code

As mentioned earlier, the shaped neighborhood "cookie cutters" are implemented in C code and run under the Sun Solaris operating system on an Enterprise 3000 workstation. The C module loads a pre-trained external weight file (thus determining the neighbor shape), then filters a set of input candidate parts with respect to a target part and similarity threshold. The module is very compact owing to the standardization of the NN topology employed through this dissertation. The C function that actually performs the feed forward pass is shown in Figure 8-5.

```
/*********************************************************/
/** function: feed_fwd(x)                                **/
/** Pass a vector through the network, return output **/
/*********************************************************/
long double feed_fwd(long double x[3])
{
    int t, u, v, n;
    long double node1[4], node2[21], node3;
    long double act1[20], act2;

    /* feed inputs to first layer */
    node1[0]  =x[0];
    node1[1]  =x[1];
    node1[2]  =x[2];
    node1[3]  =-1;
    node2[20] =-1;

    /* feed forward to hidden layer */
    for(t=0;t<20;t++)
       {
       act1[t]=0.0;
       for(u=0;u<4;u++)
          {
          act1[t]=act1[t]+(node1[u]*wt1[t][u]);
          }
       }
```

```
    /* calculate the non-linearity for hidden layer 1 */
    for(n=0;n<20;n++)
       {
       node2[n]=sigmoid(act1[n]);
       }

    /* feed forward to output layer */
    act2=0.0;
    for(v=0;v<21;v++)
       {
       act2=act2+(node2[v]*wt2[v]);
       }

    /* calculate the non-linearity for output layer */
    node3=sigmoid(act2);

    /* return the output value */
    /* return node3; */
    return act2;
}
```

Figure 8-5 C code module for NN feed forward pass

The interesting point to note with respect to this implementation is that the shaped neighborhoods are trained on an IBM 390 mainframe running MVS, yet the 121 resulting neural net weights characterizing the neighborhood are transported to the Solaris workstation environment for actual use as filters. This is made possible by the use of long double floating data types which on the Sun workstation are 16 bytes (128 bits) long. Thus the precision of the IBM mainframe is maintained.

Also, since the neural network shaped neighborhoods employ a train once, use many paradigm, the actual filtering process is just a single feed forward pass as shown in the code of Figure 8-5. This runs in milliseconds and thousands of parts can be filtered in just seconds. The entire C module, which is the generic shaped neighborhood engine, is only a few hundred lines long and is shown in Appendix B. The versatility of the method derives from the ability to load this generic engine with specific weights representing any of an infinite number of possible filters.

8.4 Results and verification

The neural network shaped neighborhoods presented in this chapter were verified against a subspace of 2011 shear tie parts returned by an SQL query. The SQL query has the following form:

```
SELECT "PART_ID" FROM PARMS.TABLE
    WHERE "BNDG_BOX_X"
    BETWEEN ("TARGET_X" - "X_OFF") AND ("TARGET_X" + "X_OFF")
    AND "BNDG_BOX_Y"
    BETWEEN ("TARGET_Y" - "Y_OFF") AND ("TARGET_Y" + "Y_OFF")
    . . .
```

This query selects the specified result set from the database. This result set is characterized as rectilinear in the sense that the boundaries are planes in the hypercube. As such, certain extremities of the subspace (the corners) contain parts which are not suitable as similar parts to a hypothetical part located at the origin of the subspace called (TARGET_X, TARGET_Y,...) in the query command. Finally, this hypercube subspace is filtered by the neural net shaped neighborhood by applying the property values for each part (corresponding to the respective point coordinates in the hypercube) to the network inputs. The neural net output is compared to a threshold to yield a selection criteria. Figure 8-6 shows 4 selected parts and Figure 8-7 shows 2 rejected parts.

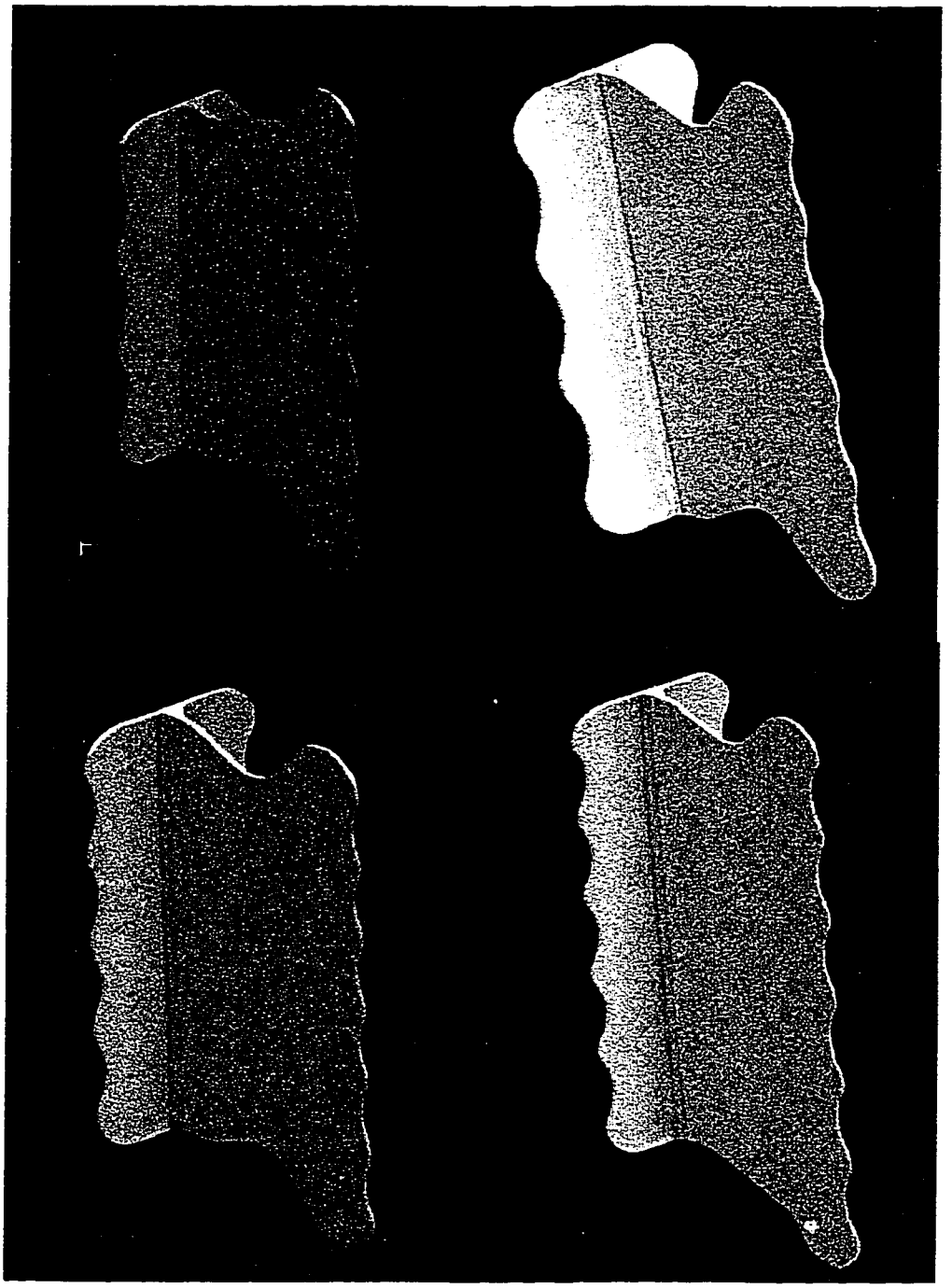


Figure 8-6. Four selected similar part candidates.

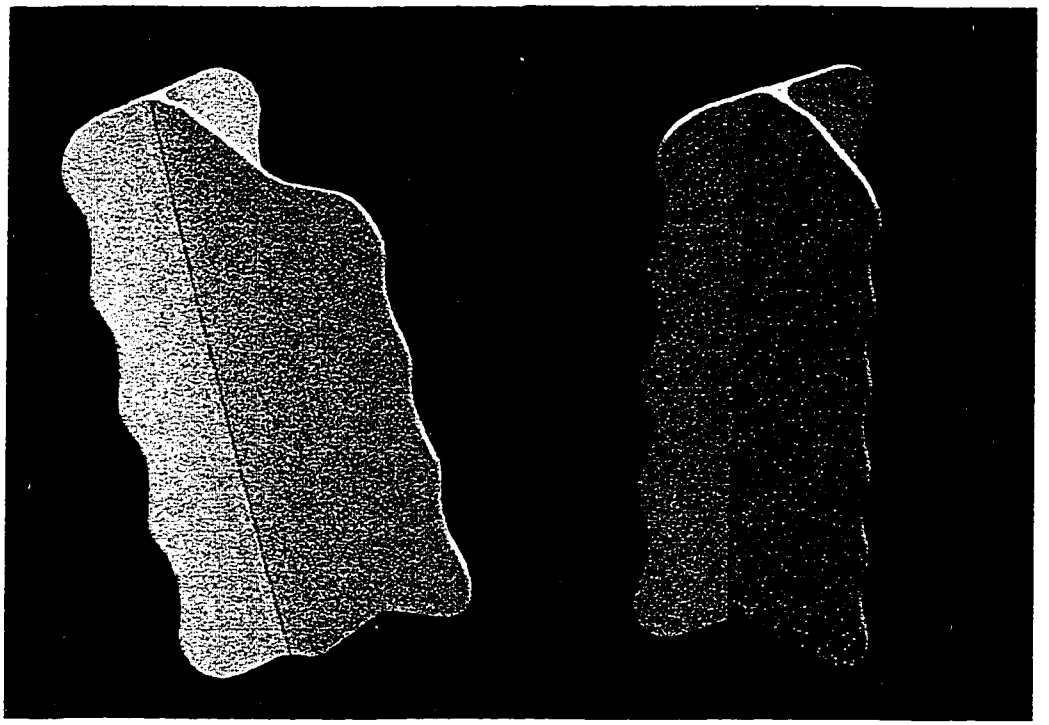


Figure 8-7. Two rejected parts.

The neural network gave results that qualitatively agree with a visual inspection of the selected and rejected parts. In this case, 2011 parts which were included in an SQL query as have relatively similar physical properties, were further reduced down to 4 parts with even greater similarity. At this point a design engineer could do a final evaluation of the 4 parts with respect to their suitability for a given design requirement.

A fuzzy logic filter was used to obtain the results shown if Figure 8-8. In this case, the given target part (top) was used to establish nominal values for the 8 part parameters. A sub-space of candidate parts was determined using these nominal values plus a standard delta. The list was narrowed to 2 candidates using a fuzzy filter defined in chapter 4.

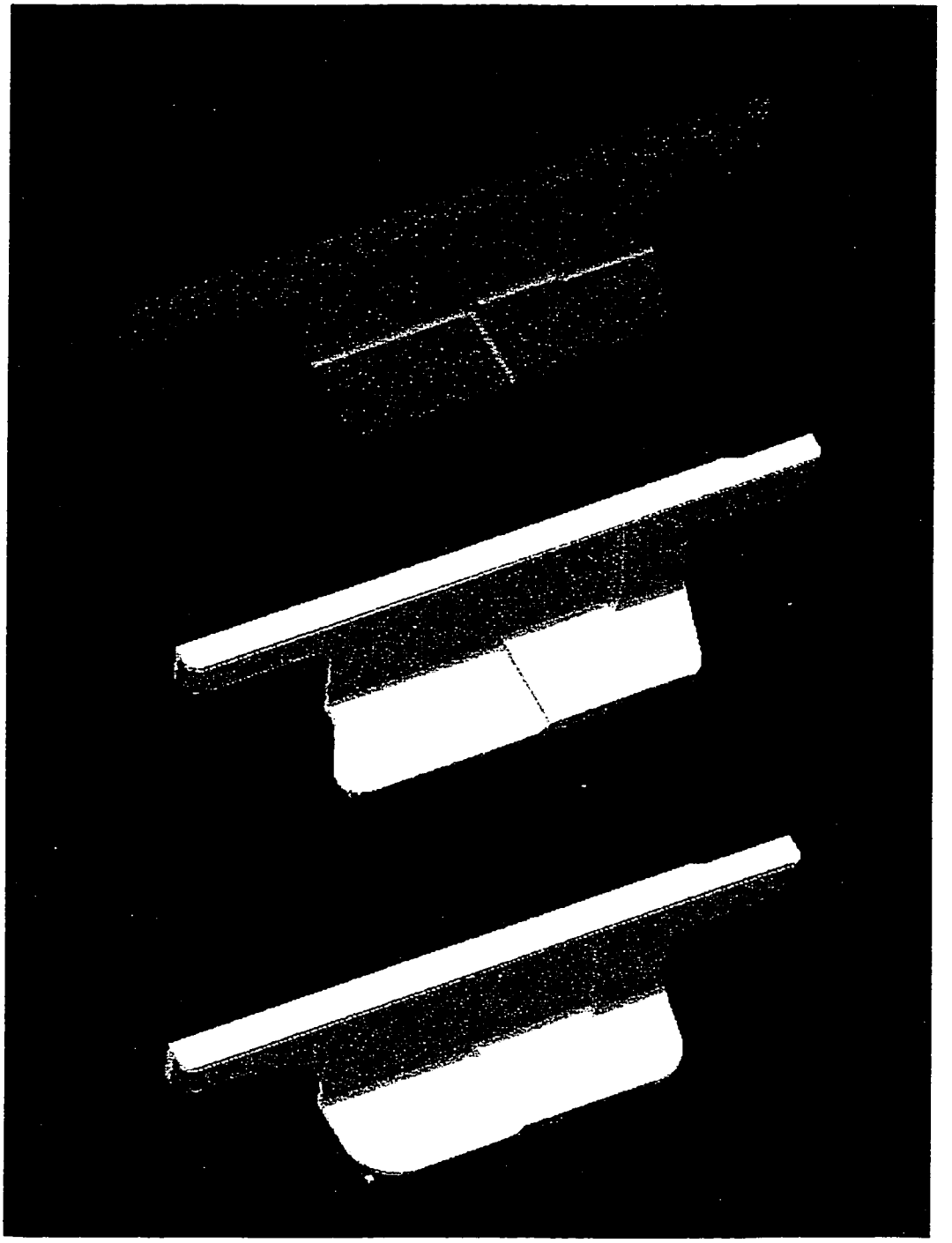


Figure 8-8. A Target part (top) and two similar parts.

Chapter 9
Conclusion And Future Work

9.1 Conclusion

Part reuse in large manufactured structures such as commercial airframes, holds tremendous opportunity for economic savings by reducing costs associated with designing, manufacturing, and maintaining individual detail parts. This opportunity can only be exploited if an effective design retrieval system is in place to identify potential reuse candidates among the millions of parts in the configuration management system. This dissertations defines shaped neighborhoods in multi-dimensional space as vehicle to facilitate such a design retrieval system. Further, this notion of shaped neighborhoods is fully developed through the use of analytic, fuzzy logic, and neural network based implementations. Results are shown for typical aerospace industry type engineering detail parts.

9.2 Future Direction

The system defined in this dissertation was created in response to a need for a large scale, definable selectivity, design retrieval system. Its features derive specifically from these requirements. One of the main features is the notion of interchangeable, stackable filters (cookie cutters) utilized to define the search criteria. The utility of this implementation is that these filters are created once and used over and over again. As such, an inventory of filters is developed suited to various specific search criteria and made available to users on demand.

Since the cookie cutters are easily stored as sets of numeric parameters associated with neural network weights, the number of cookie cutters which can be made available to application users is essentially unlimited. Future work would include the development of more and more specifically targeted filters to be added to the system inventory. Opportunity exists to define filters based on "expert knowledge". In this scenario, a knowledgeable engineer would provide filter training data by making a series of yes/no decisions regarding part similarity based on their expertise. This knowledge is captured and a new cookie cutter is created and made available to all other users. Other future work would include an interface to guide users through the cookie cutter selection process, the actual database search (filtering) process, and the ultimate visualization and selection of results.

BIBLIOGRAPHY

P. Arabshahi, J. Choi, R. Marks II, T. Audell. "Fuzzy Control of Backpropagation," IEEE International Conference on Fuzzy Systems, San Diego, California, March 1992.

P. Arabshahi, J. Choi, R. Marks II, T. Audell. "Fuzzy Parameter Adaptation in Optimization," IEEE Computational Science & Engineering, pp. 57-65, 1996.

Shumeet Baluja, "Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space," Justsystem Pittsburgh Research Center, Pittsburgh, PA. 15213.

Judith S. Bowman, Sandra L. Emerson and Marcy Darnovsky, "The Practical SQL Handbook, Using Structured Query Language," Addison-Wesley Publishing Co., Menlo Park, CA., 1993.

Gail A. Carpenter and Stephen Grossberg, "Fuzzy ART: Fast Stable Learning and Categorization of Analog Patterns by an Adaptive Resonance System," IEEE Transactions on Neural Networks, vol. 4, pp.759-771, 1991.

Don Chamberlin, "Using The NEW DB2, IBM's Object-Relational Database System," Morgan Kaufmann Publishers, Inc., San Francisco, CA., 1996.

K. W. Chan and K. L. Chan, "Multi-reference neighborhood search for vector quantization by self-organized Feature Map," Fifth International Conference on Image Processing and its Applications, Edinburgh, UK. Pp. 579-83, July 4-6, 1995.

K. W. Chan and K. L. Chan, "Multi-reference neighborhood search for vector quantization by Neural Network prediction and self-organized Feature Map," 1995 IEEE International Conference on Neural Networks Proceedings, Perth, WA, Australia. Pp. 1898-902 Vol. 4. IEEE Australia Council. Nov. 27 - Dec. 1, 1995.

B. B. Chaudhuri, "A new definition of neighborhood of a point in multi-dimensional space," Pattern Recognition Letters, Vol. 17, No. 1, pp. 11-17, Jan. 10, 1996.

Jung-Hsien Chiang. "A Hybrid Neural Network Model in Handwritten Word Recognition," Neural Networks. No. 11, pp. 337-46, 1998.

J. H. Conway and N. J. A. Sloane, "Sphere Packings, Lattices and Groups," Springer-Verlag.

Ralf Der and Gerd Balzuweit, "Constructing principal manifolds in sparse data sets by self-organizing maps with self-regulating neighborhood width," University of Leipzig, Institute of Information, Leipzig, Germany, May, 1996.

C. H. Edwards, "Advanced Calculus of Several Variables. Academic Press.

R. A. Fisher, "The Use of Multiple Measurements in Taxonomic Problems" Annual Eugenics, 7, Part II, pp 179-188 (1936)

S. Hakin. "Neural Networks: A Comprehensive Foundation," IEEE Press, Piscataway, New Jersey. 1994.

Bong-Wha-Hong, Seung-Joo-Lee, Won-Kyung-Cho. "On the Configuration of Learning Parameter to Enhance Convergence Speed of Backpropagation Neural Network"' Journal of Korean Institute of Telematics and Electronics, Vol. 338, No. 11, pp. 159-66. Nov. 1996.

Hisao Ishibuchi, Naohisa Yamamoto, Tadahiko Murata, and Hideo Tanaka, "Genetic algorithms and neighborhood search Algorithms for Fuzzy flowshop scheduling problems," Fuzzy Sets and Systems. Vol. 67, No. 1, pp. 81-100, Oct. 10, 1994.

R. A. Jacobs. "Increased Rates of Convergence Through Learning Rate Adaptation," Neural Networks, Vol. 1, 1988. Pp. 295-307.

T. Kohonen. "Self-Organization and Associative Memory," Springer, New York, 1984.

T. Kohonen, J. Kangas J. Lanksonen. "SOM_PAK: The self-organizing Map Program Package," Version 1.2. Helsinki University of Technology, Finland, 1992.

Walter Noll, "Finite-Dimensional Spaces, Algebra, Geometry, and Analysis," Martinus Nijhoff Publishers, Volume 1, pp. 39-65, 1987.

Pavel Pudil and Jana Novovicova, "Novel Methods for subset selection with respect to problem knowledge," IEEE Intelligent Systems, pp. 66-74, March/April 1998.

Kun-Chang-Lee, Imgoo-Han, Youngsig-Kwan. "Hybrid Neural Network Models for Bankruptcy Prediction," Decision Support Systems. Vol. 18, No. 1. pp. 63-72. Sept. 1996.

Tung-Hsin Su, Hsinchu Chang, Ruei-Chuan Chang and Taipei, "On constructing the relative neighborhood graphs in Euclidean k-dimensional spaces," Computing, Volume 46, No. 2, pp. 121-130, 1991

Soren Vang, "SQL and Relational Databases," Microtrend Books, San Marcos, CA., 1991.

Hua Yang and M. Palaniswami, "Convergence of self-organizing nets with high dimensional neighborhood relation," IJCNN International Joint Conference on Neural Networks. Baltimore, MD. pp. 347-351, Vol. 3, IEEE Int. Neural Network Soc. June 7-11, 1992.

Mihalis Yannakakis, "The analysis of local search problems and their heuristics", 7th Annual Symposium of Theoretical Aspects of Computer Science Proceedings. Rouen, France. pp. 298-311, 22-24 Feb. 1990.

Appendix A

Source Code for Shaped Neighborhood Visualization

```
C*************************************************************************
C                                                                        *
C    PURPOSE: RENDER A NEURAL NETWORK BASED SHAPED NEIGHBORHOOD          *
C                                                                        *
C    METHOD:  DEFINE AN IMAGE PLANE IN TERMS OF AN EYE POINT,            *
C             VIEW ANGLE, AND IMAGE RESOLUTION. CAST RAYS THROUGH        *
C             THIS IMAGE PLANE AND SOLVE FOR THE NEURAL NETWORK          *
C             INTERSECTION. REFELECT THE RAY AROUND THE SURFACE          *
C             NORMAL AT THE INTERSECTION POINT AND MEASURE THE           *
C             ANGLE BETWEEN THE REFLECTED RAY AND A LIGHT SOURCE         *
C             THE PIXEL INTENSITY IS PROPORTIONAL TO THIS ANGLE.         *
C                                                                        *
C    PROGRAMMER: F. S. HOLMAN III                                        *
C                                                                        *
C    VERSION:    1.0                                                     *
C    DATE        2/10/99                                                 *
C                                                                        *
C*************************************************************************
C
       IMPLICIT INTEGER*4    (A-Z)
C
       INTEGER*4  IMAGE(480,640), PIX
       REAL*8     DIST, LGH(3), LEN, VAL, SCALE
       REAL*8     WT1(20,4), WT2(21), X, Y
       REAL*8     FOCUS(3), CEN(3), EYEPNT(3)
       REAL*8     XRES, YRES, PIXPNT(3), HALF(3)
       REAL*8     VEC3(3), VEC4(3)
       CHARACTER*1 NXTWRD
C
       COMMON IMAGE, WT1, WT2
C
C*************************************************************************
C ***** TOP OF EXECUTABLE CODE ****************************************
      WRITE(6,2)
2     FORMAT(/,' SHADE A SHAPED NEIGHBORHOOD')
C
C OPEN NEURAL NET WEIGHTS FILE
      OPEN(UNIT=65,ERR=6000,STATUS='OLD')
C
C READ IN THE WEIGHTS DATA
8      FORMAT(F32.22)
       DO 12 J=1,20
       DO 10 K=1,4
          READ(UNIT=65,FMT=8,END=20,ERR=20)VAL
          WT1(J,K)=VAL
C         WRITE(6,*)J,K,WT1(J,K)
10     CONTINUE
12     CONTINUE
14     FORMAT('WT1(',I2,',',I2,')=',F32.28)
```

```
        DO 16 K=1,21
           READ(UNIT=65,FMT=8,END=20,ERR=20)VAL
           WT2(K)=VAL
16      CONTINUE
19      FORMAT('WT2(',I2,')=',F32.28)
        GOTO 100
C
20      CONTINUE
        WRITE(6,*)'*** ERROR IN WEIGHTS READ ROUTINE ***'
        GOTO 6000
C
C*************************************************************
C OPEN OUTPUT FILE
100    CONTINUE
       OPEN(UNIT=50,ERR=6000,STATUS='NEW')
       OPEN(UNIT=60,ERR=6000,STATUS='OLD')
C
C INITIALIZE WRITE POINTER
       RDPNT=81
       WTPNT=1
       ERRCNT=0
C
C INITIALIZE THE IMAGE ARRAY
        DO 210 J=1,480
        DO 200 K=1,640
           IMAGE(J,K)=0
200     CONTINUE
210     CONTINUE
C
C DEFINE THE LIGHT SOURCE
        LGH(1)=10.0
        LGH(2)=15.0
        LGH(3)=15.0
        LEN=DSQRT((LGH(1)**2)+(LGH(2)**2)+(LGH(3)**2))
        LGH(1)=LGH(1)/LEN
        LGH(2)=LGH(1)/LEN
        LGH(3)=LGH(1)/LEN
        WRITE(6,*)'LIGHT=',LGH
C
C SET THE FOCUS POINT
        FOCUS(1)=5.0
        FOCUS(2)=5.0
        FOCUS(3)=5.0
        WRITE(6,*)'FOCUS=',FOCUS
C
C DEFINE THE EYE POINT
        READ(5,50)EYEPNT(1)
        READ(5,50)EYEPNT(2)
        READ(5,50)EYEPNT(3)
50      FORMAT(F8.4)
C       EYEPNT(1)=8.0
C       EYEPNT(2)=8.0
C       EYEPNT(3)=20.0
```

```
C GET VIEW ANGLE AND IMAGE PLANE
58      CONTINUE
        CEN(1)=EYEPNT(1)-FOCUS(1)
        CEN(2)=EYEPNT(2)-FOCUS(2)
        CEN(3)=EYEPNT(3)-FOCUS(3)
C       WRITE(6,*)'CEN  =',CEN
C       WRITE(6,*)'CALLING GETTWO'
        CALL GETTWO(CEN,VEC3,VEC4)
        WRITE(6,*)'VEC3=',VEC3
        WRITE(6,*)'VEC4=',VEC4
C
C SPLIT THE DIFFERENCE TO EYEPOINT
        HALF(1)=FOCUS(1)+((EYEPNT(1)-FOCUS(1))/2.0)
        HALF(2)=FOCUS(2)+((EYEPNT(2)-FOCUS(2))/2.0)
        HALF(3)=FOCUS(3)+((EYEPNT(3)-FOCUS(3))/2.0)
        WRITE(6,*)'HALF WAY =',HALF
C
C*************************************************************
C DOUBLE LOOP TO CREATE IMAGE
        SCALE=125.0
        XRES=DFLOAT(480)/(2.0*SCALE)
        YRES=DFLOAT(640)/(2.0*SCALE)
        DO 1000 U=1,480
        WRITE(6,*)'SCANLINE = ',U
        DO 990 V=1,640
C          WRITE(6,*)'***************** ',V,' ***************************'
           X=(DFLOAT(481-U)/SCALE)
           Y=(DFLOAT(641-V)/SCALE)
C
C CALCULATE PIXEL PLANE POINT
        PIXPNT(1)=HALF(1)+(VEC3(1)*(X-XRES))+(VEC4(1)*(Y-YRES))
        PIXPNT(2)=HALF(2)+(VEC3(2)*(X-XRES))+(VEC4(2)*(Y-YRES))
        PIXPNT(3)=HALF(3)+(VEC3(3)*(X-XRES))+(VEC4(3)*(Y-YRES))
C
C FIND THE INTERSECTION WITH THE NEURAL NET
           CALL FUNC(EYEPNT,PIXPNT,LGH,VEC3,VEC4,PIX,ERRCNT)
           IMAGE(U,V)=PIX
990     CONTINUE
1000    CONTINUE
C PRINT OUT ERROR COUNT
        WRITE(6,*)'ERROR COUNT = ',ERRCNT
C
C*************************************************************
C CREATE THE TIFF IMAGE
        DO 1500 Q=1,256
           CALL RDBUF(ISCAN,NXTWRD,RDPNT)
           IF(RDPNT.EQ.0) GOTO 6000
           CALL WTBUF(ISCAN,WTPNT)
1500    CONTINUE
C
        WRITE(6,*)'WRITE IMAGE TO TIFF FILE'
        DO 2010 L=1,480
        DO 2000 M=1,640
```

```
            J=L
            K=641-M
            PIX=IMAGE(J,K)
            IF(PIX.EQ.0) THEN
               CALL WTBUF(20,WTPNT)
               CALL WTBUF(20,WTPNT)
               PIX=(128+(J/8))-1
               CALL WTBUF(PIX,WTPNT)
            ELSE
C           ** RED
               CALL WTBUF(PIX,WTPNT)
C           ** GREEN
               CALL WTBUF(PIX,WTPNT)
C           ** BLUE
               CALL WTBUF(PIX,WTPNT)
            ENDIF
2000    CONTINUE
2010    CONTINUE
C
C FLUSH THE WRITE BUFFER
        DO 2020 K=1,80
              CALL WTBUF(0,WTPNT)
2020    CONTINUE
C
C*************************************************************
6000    CONTINUE
        STOP
        END
C
C*************************************************************
      SUBROUTINE WTBUF(OUT,WTPNT)
C*************************************************************
C  WRITE WORD TO BUFFER, WRITE BUFFER WHEN NECESSARY
C*************************************************************
      IMPLICIT INTEGER*4 (A-Z)
      INTEGER*4 WTPNT,OUT,OUTWRD,SAV
      CHARACTER*1 NXTWRD(4), WBUFF(80)
      EQUIVALENCE (OUTWRD,NXTWRD(1))
C
50    CONTINUE
C  CHECK FOR END OF BUFFER
      IF (WTPNT.GT.80) GOTO 100
      OUTWRD=IOR(OUT,SAV)
      WBUFF(WTPNT)=NXTWRD(4)
      WTPNT=WTPNT+1
      GOTO 200
C
100   CONTINUE
C  BUFFER FULL SO WRITE NEXT RECORD
      WRITE(UNIT=50,FMT=120,ERR=130)WBUFF
120   FORMAT(80A1)
      WTPNT=1
      GOTO 50
```

```
130    WTPNT=0
200    CONTINUE
       RETURN
       END
C
C*************************************************************
       SUBROUTINE RDBUF(ISCAN,NXTWRD,RDPNT)
C*************************************************************
C  RETURN NEXT WORD, READ FROM FILE WHEN NECESSARY
C*************************************************************
C
       IMPLICIT INTEGER*4 (A-Z)
       INTEGER*4 RDPNT,ISCAN
       CHARACTER*1 NXTWRD,RBUFF(80)
C
50     CONTINUE
C  CHECK FOR END OF BUFFER
       IF (RDPNT.GT.80) GOTO 100
       NXTWRD=RBUFF(RDPNT)
       RDPNT=RDPNT+1
       GOTO 200
C
100    CONTINUE
C  BUFFER EMPTY SO READ NEXT RECORD
       READ(UNIT=60,FMT=120,END=130,ERR=130)RBUFF
120    FORMAT(80A1)
       RDPNT=1
       GOTO 50
C
130    RDPNT=0
200    CONTINUE
       ISCAN=ICHAR(NXTWRD)
       RETURN
       END
C
C*************************************************************
       SUBROUTINE FUNC(STAR,END,LGH,VEC3,VEC4,VALUE,ERRCNT)
C*************************************************************
C  SET UP A RAY AND FIND FUNCTION INTERSECTION
C*************************************************************
       IMPLICIT INTEGER*4 (A-Z)
       REAL*8     X, Y, STAR(3), END(3), ZPNT(3), OFFSET
       REAL*8     NRM(3), MAG, THETAR, THETA, LGH(3)
       REAL*8     LEN, WPNT(3), XPNT(3), VEC1(3), VEC2(3)
       REAL*8     OFF1(3), OFF2(3), VEC3(3), VEC4(3)
       INTEGER*4 VALUE, ERRCNT
C
C GET THE INTERSECTION
C      WRITE(6,*)'************ ENTERED FUNC ************'
C      WRITE(6,10)STAR,END
10     FORMAT(' STAR=',3F10.4,' END=',3F10.4)
       CALL EVAL(STAR,END,ZPNT,MAG,ERRCNT,GOTPIX)
       VALUE=0
```

```
      IF(GOTPIX.NE.1) GOTO 1000
C
C GET FIRST NEARBY POINT
      OFF1(1)=END(1)+(VEC3(1)*0.02)+(VEC4(1)*0.0)
      OFF1(2)=END(2)+(VEC3(2)*0.02)+(VEC4(2)*0.0)
      OFF1(3)=END(3)+(VEC3(3)*0.02)+(VEC4(3)*0.0)
C     WRITE(6,*)'************************'
C     WRITE(6,*)'END =',END
C     WRITE(6,*)'OFF1=',OFF1
      CALL EVAL(STAR,OFF1,WPNT,MAG,ERRCNT,GOTPIX1)
C
C GET SECOND NEARBY POINT
      OFF2(1)=END(1)+(VEC3(1)*0.0)+(VEC4(1)*0.02)
      OFF2(2)=END(2)+(VEC3(2)*0.0)+(VEC4(2)*0.02)
      OFF2(3)=END(3)+(VEC3(3)*0.0)+(VEC4(3)*0.02)
C     WRITE(6,*)'************************'
C     WRITE(6,*)'END =',END
C     WRITE(6,*)'OFF2=',OFF2
      CALL EVAL(STAR,OFF2,XPNT,MAG,ERRCNT,GOTPIX2)
C
C SKIP IF NO INTERSECTION
      IF((GOTPIX1.NE.1).OR.(GOTPIX2.NE.1)) GOTO 1000
C     WRITE(6,*)'WPNT=',WPNT
C     WRITE(6,*)'XPNT=',XPNT
C
C CREATE 2 VECTORS
      DO 800 T=1,3
         VEC1(T)=WPNT(T)-ZPNT(T)
         VEC2(T)=XPNT(T)-ZPNT(T)
800   CONTINUE
C
C GET NORMAL (CROSS PRODUCT)
      NRM(1)=((VEC1(2)*VEC2(3))-(VEC2(2)*VEC1(3)))
      NRM(2)=-((VEC1(1)*VEC2(3))-(VEC2(1)*VEC1(3)))
      NRM(3)=((VEC1(1)*VEC2(2))-(VEC2(1)*VEC1(2)))
C
C GET THE ANGLE BETWEEN VECTORS
      LEN=DSQRT((NRM(1)**2)+(NRM(2)**2)+(NRM(3)**2))
C     WRITE(6,*)'NRM=',NRM,' LEN=',LEN
      NRM(1)=NRM(1)/LEN
      NRM(2)=NRM(2)/LEN
      NRM(3)=NRM(3)/LEN
      THETAR=DACOS((NRM(1)*LGH(1))+(NRM(2)*LGH(2))+
     &                  (NRM(3)*LGH(3)))
      THETA=(57.29578*THETAR)
      VALUE=IDNINT(THETA)*2
C     WRITE(6,*)'PIX=',VALUE
C
1000  CONTINUE
      RETURN
      END
C
```

```
C*************************************************************
      SUBROUTINE EVAL(STAR,END,ZPNT,MAG,ERRCNT,GOTPIX)
C*************************************************************
C  GET INTERSECTION OF VECTOR AND FUNCTION
C*************************************************************
C
      IMPLICIT INTEGER*4 (A-Z)
C
      REAL*8 STAR(3),END(3), ZPNT(3), RAY(3), ALPHA
      REAL*8 BEFORE, MAG, DIF, PUSH, LEN
      REAL*8 SEP1, SEP2, SEP3, SEPTOT
      INTEGER*4  ERRCNT, GOTPIX
C
C     WRITE(6,*)'********* GET FUNCTION ************'
      GOTPIX=0
      PUSH=-0.002
      ALPHA=1.1
      ZPNT(1)=STAR(1)
      ZPNT(2)=STAR(2)
      ZPNT(3)=STAR(3)
C     WRITE(6,10)ZPNT,END
10    FORMAT(' ZPNT=',3F10.4,' END=',3F10.4)
C
C GET THE UNIT RAY
      RAY(1)=END(1)-STAR(1)
      RAY(2)=END(2)-STAR(2)
      RAY(3)=END(3)-STAR(3)
      LEN=DSQRT((RAY(1)**2)+(RAY(2)**2)+(RAY(3)**2))
      RAY(1)=RAY(1)/LEN
      RAY(2)=RAY(2)/LEN
      RAY(3)=RAY(3)/LEN
C
C LOOP UNTIL CLOSE
      J=0
100   CONTINUE
         MAG=0.0
         J=J+1
C        WRITE(6,*)'********** LOOP',J,' **************'
         IF(J.GT.300) THEN
C           WRITE(6,*)'***DIVERGING***'
            ERRCNT=ERRCNT+1
            DO 125 Q=1,3
               ZPNT(Q)=0.0
               MAG=0.0
125         CONTINUE
            GOTO 200
         ENDIF
C
C CHECK FOR OUT OF RANGE
         SEP1=(STAR(1)-ZPNT(1))**2
         SEP2=(STAR(2)-ZPNT(2))**2
         SEP3=(STAR(3)-ZPNT(3))**2
         SEPTOT=DSQRT(SEP1+SEP2+SEP3)
```

```
          IF(SEPTOT.GT.(18.0)) THEN
C             WRITE(6,*)'***OUT OF RANGE***'
              ERRCNT=ERRCNT+1
              DO 127 Q=1,3
                 ZPNT(Q)=0.0
                 MAG=0.0
127           CONTINUE
              GOTO 200
          ENDIF
C
C COMPARE NN Z VALUE TO CURRENT Z VALUE
          CALL EV(ZPNT,MAG)
C         WRITE(6,*)'MAG=',MAG
          DIF=MAG-ALPHA
C         WRITE(6,*)'DIF  =',DIF
          IF(DABS(DIF).LT.(0.005)) THEN
             GOTPIX=1
C            WRITE(6,*)'*** MATCH ***'
C            WRITE(6,*)' '
             GOTO 200
          ENDIF
C
C PUSH ZPNT ALONG VECTOR
          ZPNT(1)=ZPNT(1)+(RAY(1)*(PUSH*(300-J)*DIF))
          ZPNT(2)=ZPNT(2)+(RAY(2)*(PUSH*(300-J)*DIF))
          ZPNT(3)=ZPNT(3)+(RAY(3)*(PUSH*(300-J)*DIF))
C         WRITE(6,*)'ZPNT =',ZPNT
          GOTO 100
C
200    CONTINUE
       RETURN
       END
C
C*************************************************************
       SUBROUTINE EV(ZPNT,MAG)
C*************************************************************
C  EVALUATE THE SPACE POINT
C*************************************************************
       IMPLICIT INTEGER*4 (A-Z)
C
       INTEGER*4  IMAGE(480,640)
       REAL*8     ZPNT(3), MAG
       REAL*8     WT1(20,4), WT2(21)
       REAL*8     NODE1(4), NODE2(21), NODE3, SIGPRIME
       REAL*8     ACT1(20), ACT2, ERR, LRP
       COMMON IMAGE, WT1, WT2
C
C FEED FORWARD TO NODE 1
      NODE1(1)  = ZPNT(1)
      NODE1(2)  = ZPNT(2)
      NODE1(3)  = ZPNT(3)
      NODE1(4)  = -1.0
      NODE2(21) = -1.0
```

```
C*****************************************
C FEED FORWARD TO HIDDEN LAYER 1
      DO 150 T=1,20
         ACT1(T)=0.0
         DO 140 U=1,4
            ACT1(T)=ACT1(T)+(NODE1(U)*WT1(T,U))
140      CONTINUE
150   CONTINUE
C     WRITE(6,*)'ACT1=',ACT1
C
C CALCULATE THE NON-LINEARITIES FOR HIDDEN LAYER 1
      DO 160 N=1,20
         CALL SIGMOID(ACT1(N),NODE2(N))
160   CONTINUE
C     WRITE(6,*)'NODE2=',NODE2
C
C*****************************************
C FEED FORWARD TO OUTPUT LAYER
      ACT2=0.0
      DO 170 V=1,21
         ACT2=ACT2+(NODE2(V)*WT2(V))
170   CONTINUE
C     WRITE(6,*)' '
C     WRITE(6,*)'************************************************'
C     WRITE(6,*)' POINT =',ZPNT
C
C CALCULATE THE NON-LINEARITIES FOR OUTPUT LAYER
      DO 185 N=1,3
         CALL SIGMOID(ACT2,NODE3)
185   CONTINUE
C
C RETURN THE MAGNITUDE
      MAG=ACT2
C
      RETURN
      END
C
C*************************************************************************
       SUBROUTINE SIGMOID(VALIN,VALOUT)
C*************************************************************************
C  COMPUTE THE SIGMOID OF A NUMBER
C*************************************************************************
       IMPLICIT INTEGER*4   (A-Z)
       REAL*8   VALIN, VALOUT
C
C CALCULATE THE NON-LINEARITIES
         IF(ABS(VALIN).GT.(40.0)) THEN
            IF(VALIN.LT.(0.0)) VALOUT=0.0
            IF(VALIN.GT.(0.0)) VALOUT=1.0
            IF(VALIN.EQ.(0.0)) VALOUT=0.5
         ELSE
            VALOUT=1.0/(1.0+EXP(-VALIN))
         ENDIF
```

```
1000    CONTINUE
        RETURN
        END
200    CONTINUE
       RETURN
       END
C
C*************************************************************
       SUBROUTINE GETTWO(CEN,VEC1,VEC2)
C*************************************************************
C  GET TWO NORMAL VECTORS TO A UNIT VECTOR
C*************************************************************
       IMPLICIT INTEGER*4 (A-Z)
C
       REAL*8 CEN(3), VEC1(3), VEC2(3)
       REAL*8 TMP1(3), TMP2(3), TMP3(3), TMP4(3), TMP5(3)
       REAL*8 GAMMAR, THETAR, MAG, SIGN, RAD
C
C      WRITE(6,*)'********* ENTERED GETTWO **********'
       SIGN=1.0
C
       MAG=DSQRT((CEN(1)**2)+(CEN(2)**2)+(CEN(3)**2))
       CEN(1)=CEN(1)/MAG
       CEN(2)=CEN(2)/MAG
       CEN(3)=CEN(3)/MAG
C      WRITE(6,10)CEN
10     FORMAT(' CENTER=',3F12.8)
C
C GET THETA IN RADIANS
       RAD=DSQRT((CEN(1)**2)+(CEN(2)**2))
C      WRITE(6,*)'ANGLE IN RADIANS = ',RAD
       THETAR=DACOS(CEN(1)/RAD)
C      WRITE(6,*)'THETAR = ',THETAR
       CALL ROTZ(CEN,THETAR,TMP1)
       IF(TMP1(2).GT.(0.001)) THEN
          CALL ROTZ(CEN,-THETAR,TMP1)
          SIGN=-1.0
       ENDIF
C
       RAD=DSQRT((TMP1(1)**2)+(TMP1(3)**2))
       GAMMAR=DACOS(TMP1(1)/RAD)
C      WRITE(6,*)'GAMMAR = ',GAMMAR
C
       TMP2(1)=0.0
       TMP2(2)=0.0
       TMP2(3)=1.0
       CALL ROTY(TMP2,-GAMMAR,TMP3)
       CALL ROTZ(TMP3,(-SIGN*THETAR),VEC1)
C
       TMP4(1)=0.0
       TMP4(2)=1.0
       TMP4(3)=0.0
       CALL ROTZ(TMP4,(-SIGN*THETAR),VEC2)
```

```
C     WRITE(6,*)'TMP1=',TMP1
C     WRITE(6,*)'TMP2=',TMP2
C     WRITE(6,*)'TMP3=',TMP3
C     WRITE(6,*)'TMP4=',TMP4
C     WRITE(6,*)'VEC1=',VEC1
C     WRITE(6,*)'VEC2=',VEC2
C
100   CONTINUE
      RETURN
      END
C
C*************************************************************
      SUBROUTINE ROTX(WIN,ANG,WOUT)
C*************************************************************
C  ROTATE AROUND X AXIS
C*************************************************************
      IMPLICIT INTEGER*4 (A-Z)
C
      REAL*8 WIN(3), ANG, WOUT(3)
C
C DO THE ROTATION TRANSFORMATION
      WOUT(1)=WIN(1)
      WOUT(2)=((WIN(2)*DCOS(ANG))-(WIN(3)*DSIN(ANG)))
      WOUT(3)=((WIN(2)*DSIN(ANG))+(WIN(3)*DCOS(ANG)))
C
      RETURN
      END
C
C*************************************************************
      SUBROUTINE ROTY(WIN,ANG,WOUT)
C*************************************************************
C  ROTATE AROUND Y AXIS
C*************************************************************
      IMPLICIT INTEGER*4 (A-Z)
C
      REAL*8 WIN(3), ANG, WOUT(3)
C
C DO THE ROTATION TRANSFORMATION
      WOUT(1)=((WIN(1)*DCOS(ANG))+(WIN(3)*DSIN(ANG)))
      WOUT(2)=WIN(2)
      WOUT(3)=(-(WIN(1)*DSIN(ANG))+(WIN(3)*DCOS(ANG)))
C
      RETURN
      END
C
C*************************************************************
      SUBROUTINE ROTZ(WIN,ANG,WOUT)
C*************************************************************
C  ROTATE AROUND Z AXIS
C*************************************************************
      IMPLICIT INTEGER*4 (A-Z)
      REAL*8 WIN(3), ANG, WOUT(3)
```

```
C
C DO THE ROTATION TRANSFORMATION
      WOUT(1)=((WIN(1)*DCOS(ANG))-(WIN(2)*DSIN(ANG)))
      WOUT(2)=((WIN(1)*DSIN(ANG))+(WIN(2)*DCOS(ANG)))
      WOUT(3)=WIN(3)
C
      RETURN
      END
```

Appendix B

C Code for Shaped Neighborhood Evaluation

```
/*********************************************************
*                                                        *
*       Shaped Neighborhood: Cookie Cutter               *
*       Programmer : Frank Holman                        *
*                                                        *
*********************************************************/

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

/* Function prototype */
long double feed_fwd(long double x[3]);
long double sigmoid(long double in_val);

/* Global Declarations */
int  DBG=2;
long double wt1[20][4], wt2[21];


main(argc, argv)
int argc;
char *argv[];

{
     FILE *fp_in, *fp_out, *fp_targ, *fp_wgt, *fopen();

     int  i, j, n, id, part_id[1000], prtcnt=0;
     long double val, output, scale=2.0;
     long double cand[3], targ[3], x[3];
     long double threshold= 1.0000;

/* Print out prolog */
        printf("\n");
        printf("  ********************************************\n");
        printf("  *                                          *\n");
        printf("  *             Shaped Neighborhood          *\n");
        printf("  *                                          *\n");
        printf("  *  Load a Neural Network from the external *\n");
        printf("  *  weights file. Read data from an input   *\n");
        printf("  *  file, and filter it with respect to a   *\n");
        printf("  *  target point and NN threshold parameter *\n");
        printf("  *                                          *\n");
        printf("  ********************************************\n");
        printf("\n");

/* Print out the command line parameters */
     if(DBG>1) printf("argv[1] = %s, argc = %d\n", argv[1],argc);
```

```
/**** Open the weights file, input, and output files ***************/
      if (argc == 5)
        {
        /* Open the weights file */
        if((fp_wgt = fopen(argv[1],"r")) == NULL)
          {
          printf("can't open %s\n", argv[1]);
          exit(1);
          }
        /* Open the input file */
        if((fp_in  = fopen(argv[2],"r")) == NULL)
          {
          printf("can't open %s\n", argv[2]);
          exit(1);
          }
        /* Open the target file */
        if((fp_targ = fopen(argv[3],"r")) == NULL)
          {
          printf("can't open %s\n", argv[3]);
          exit(1);
          }
        /* Open the output file */
        if((fp_out = fopen(argv[4],"w")) == NULL)
          {
          printf("can't open %s\n", argv[4]);
          exit(1);
          }
        }
      else
        {
        /* Prompt for correct syntax */
        printf("syntax: shape weight.file in.file target.file output.file\n");
        exit(1);
        }

/**** Read in external weights ***********************************/

      if(DBG>2) printf("\n**** Read in the Weights ****\n");
      /* Input layer */
      for (i=0;i<20;i++)
        {
        for (j=0;j<4;j++)
          {
          fscanf(fp_wgt,"%Lff",&val);
          wt1[i][j]=val;
          if(DBG>2) printf("wt1[%d][%d]=%Lff\n",i,j,wt1[i][j]);
          }
        }
```

```
    /* Output layer */
    for (i=0;i<21;i++)
      {
      fscanf(fp_wgt,"%Lf",&val);
      wt2[i]=val;
      if(DBG>2) printf("wt2[%d]=%f\n",i,wt2[i]);
      }


/**** Read in Target data ******************************************/

    printf("\n**** Read target file ****\n");
    n=fscanf(fp_targ,"%Lf" "%Lf" "%Lf",&targ[0],&targ[1],&targ[2]);
    if(DBG>1) printf("Target = %Lf %Lf %Lf\n",targ[0],targ[1],targ[2]);


/**** Pass input data through neural network ***********************/

    printf("\n**** Loop over input data ****\n");
    if(DBG>1) printf("\n");
    while(n>2)
      {
      /* read input data */
      n=fscanf(fp_in,"%d %Lf" "%Lf" "%Lf",&id,&cand[0],&cand[1],&cand[2]);

      printf("\n********* Part_id = %d **************\n",id);
      if(DBG>1) printf("Target    = %Lf %Lf %Lf\n",targ[0],targ[1],targ[2]);
      if(DBG>1) printf("Candidate = %Lf %Lf %Lf\n",cand[0],cand[1],cand[2]);

      /* translate to origin and scale */
      for (i=0;i<3;i++)
        {
        x[i]=((cand[i]-targ[i]) * scale)+5.0;
        }

      /* send values through network */
      if(DBG>2) printf("**** send data to Neural Network ****\n");
      output=feed_fwd(x);
      if(DBG>1) printf("output=%Lf\n",output);
      if(output>threshold)
        {
        printf("*** MATCH ***\n");
        fprintf(fp_out,"\n**** Part ID = %d ********************************\n",id);
        fprintf(fp_out,"Target    = %Lf %Lf %Lf\n",targ[0],targ[1],targ[2]);
        fprintf(fp_out,"Candidate = %Lf %Lf %Lf\n",cand[0],cand[1],cand[2]);
        fprintf(fp_out,"output=%Lf\n",output);
        }
      }
```

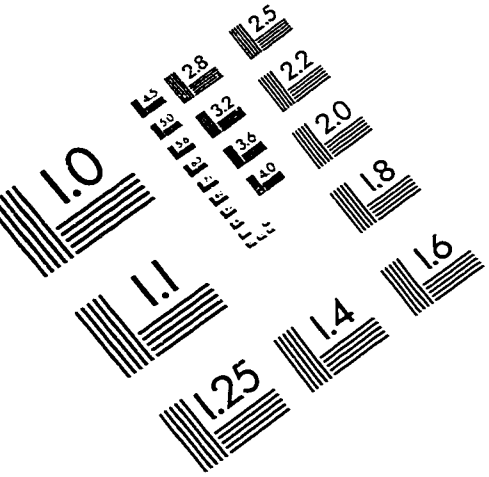
```
/*********************************************************************/

/* Close the files. */
      fclose(fp_in);
      fclose(fp_out);
      fclose(fp_targ);
      fclose(fp_wgt);

/* Normal exit */
      puts("\n**** Normal program termination ****\n");

      exit(0);
}
/*******************************************************/
/** function: feed_fwd(x)                             **/
/** Pass a vector through the network, return output **/
/*******************************************************/
long double feed_fwd(long double x[3])
{
      int t, u, v, n;
      long double node1[4], node2[21], node3;
      long double act1[20], act2;

      /* feed inputs to first layer */
      node1[0] =x[0];
      node1[1] =x[1];
      node1[2] =x[2];
      node1[3] =-1;
      node2[20] =-1;

      /* feed forward to hidden layer */
      for(t=0;t<20;t++)
        {
        act1[t]=0.0;
        for(u=0;u<4;u++)
          {
          act1[t]=act1[t]+(node1[u]*wt1[t][u]);
          }
        }

      /* calculate the non-linearities for hidden layer 1 */
      for(n=0;n<20;n++)
        {
        node2[n]=sigmoid(act1[n]);
        }

      /* feed forward to output layer */
      act2=0.0;
      for(v=0;v<21;v++)
        {
        act2=act2+(node2[v]*wt2[v]);
        }
```

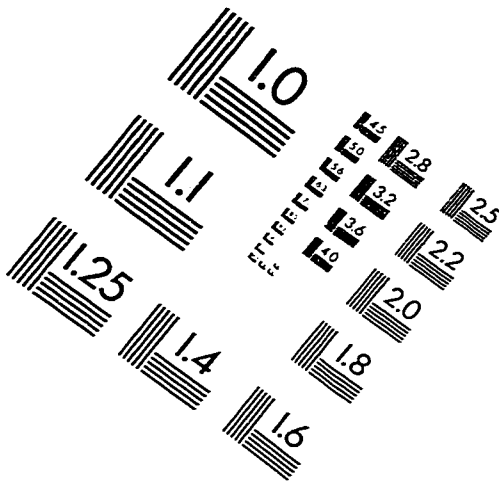
```
    /* calculate the non-linearity for output layer */
    node3=sigmoid(act2);

    /* return the output value */
    /* return node3; */
    return act2;
}

/*******************************************************/
/** function: sigmoid(val_in)                         **/
/** Return the sigmoid of an input value              **/
/*******************************************************/
long double sigmoid(long double val_in)
{
    long double val_out;

    if(abs(val_in)>40.0)
      {
      if(val_in>0.0) val_out=1.0;
      if(val_in<0.0) val_out=0.0;
      }
    else
      {
      val_out=1.0/(1.0+exp(-val_in));
      }

   /* Return sigmoid */
   return val_out;
}
```

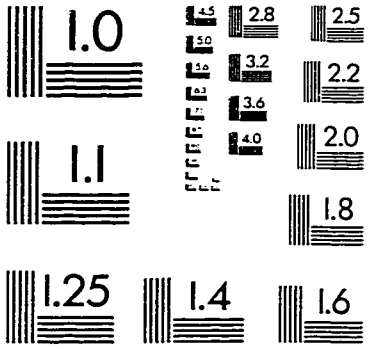
VITA

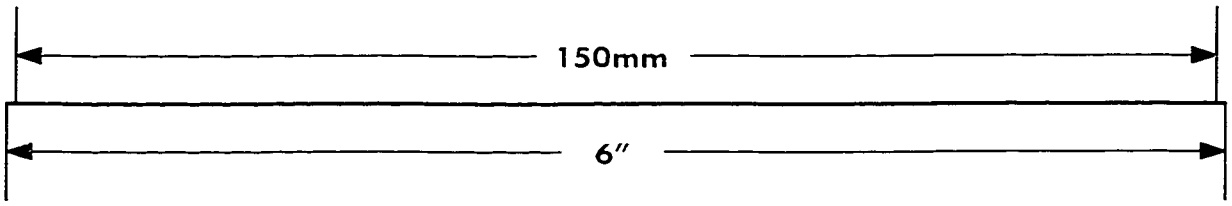
Frank Holman graduated from Lakeside High School, Seattle, Washington in June of 1967. He attended the University of Washington starting in 1970. He graduated with a Bachelor of Science degree in Oceanography in June of 1979. In June of 1994, he joined the Department of Electrical Engineering at the University of Washington and received a Master of Science degree in Electrical Engineering in June of 1996. He was advanced to Ph.D. candidacy summer quarter of 1998.

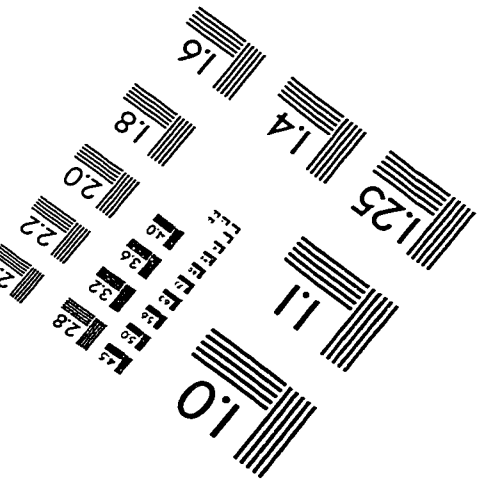
IMAGE EVALUATION TEST TARGET (QA-3)

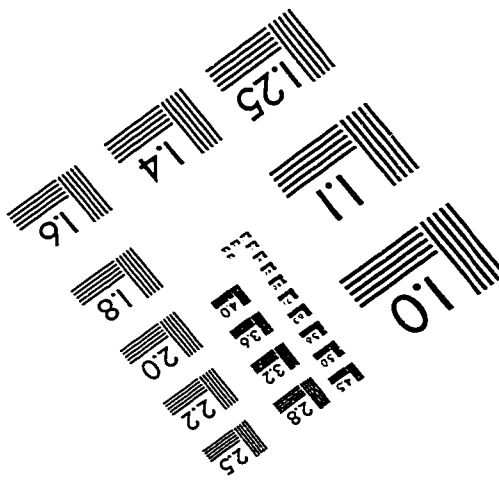












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989