2001

Statistical modeling and design for CMM-type data locating known two-dimensional geometries

Dewi Rahardja
Iowa State University

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.

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.

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

Statistical modeling and design for CMM-type data locating known two-dimensional geometries

by

Dewi Rahardja

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Industrial Engineering

Major Professor: Stephen B.Vardeman

Iowa State University

Ames, Iowa

2001

UMI Number: 3016743

UMI Microform 3016743

Bell & Howell Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

Graduate College
Iowa State University

This is to certify that the Doctoral dissertation of

Dewi Rahardja

has met the dissertation requirements of Iowa State University

Signature was redacted for privacy.

Major Professor

Signature was redacted for privacy.

For the Major Program

Signature was redacted for privacy.

For the Graduate College

TABLE OF CONTENTS

Statistical modeling and design for CMM-type data locating known two-dimensional geometries

Dewi Rahardja

Major Professor: Stephen B.Vardeman
Iowa State University

This research was motivated by a problem from a car-manufacturer that needed to compare/analyze two hood assembly fixtures. The consistency of the car hood placement was examined for both fixtures. A "constant width" gap between the hood and the car's body is preferred. A "wide" gap at one point and a "narrow" gap at another constitutes a bad quality placement of the car hood, and will affect customer satisfaction.

3-D data were taken using a Coordinate Measuring Machine (CMM) with 12 (fixed) probe paths to the hood. For each fixture, ten "12-dimensional" vectors were taken and the original goal was to compare the 2 fixtures and decide which one gives more consistent/better placement of the hood. As a first step in statistical modeling and analysis for this type of problem, we consider a 2-dimensional idealization ignoring the 4 "Up-Down" measurements and replacing the complicated real hood geometry with simple ideal geometries of approximately the same overall size. Then the data become ten "8-dimensional" vectors.

A "rough" analysis done by the car-manufacturer analyst was to take ten "8-dimensional" vectors for both fixtures A and B and make 8 comparisons of sample variances (one probe path at a time). This ignores geometry effects/physical dependence and answers the wrong question (how variable are the measurements, versus how variable is hood placement).

In this dissertation, we consider statistical analysis specifically focused on the issue of "Where is the hood?" (as opposed to "What is the distribution of a Y_i ?"). Our main contribution is in the realm of study planning, where the object is to choose a set of probe paths that provide optimal precision for estimating the position of a single "hood" placement. We propose a figure of merit for comparing alternative designs (data collection plans) and compare several algorithms for optimizing this criterion. Our comparisons of algorithms are across design sizes, "hood" geometries and nominal locations. We conclude that the design sizes, geometries, and nominal locations affect the optimum design while the candidate probe paths doesn't. The paths in an optimum/best design will generally be located closest to the "extremes" of the object boundary or will be aimed near "corners" of the object.

CHAPTER 1. INTRODUCTION

This work was motivated by a problem from a car-manufacturer that needed to compare/analyze two hood fixtures. Using both fixtures, placement of car hoods was performed and the consistency of the placement was examined. A "constant width" gap between the hood and the car's body is preferred. A "wide" gap at one point and a "narrow" gap at another constitutes a bad quality placement of the car hood, and will affect customer satisfaction.

Data were obtained using a Coordinate Measuring Machine (CMM) as shown in schematic form on Figure 1.1. Indicated on the figure are 12 (fixed) probe paths to the hood. A datum for any one path is a measurement of where along the path the probe first touches the hood.

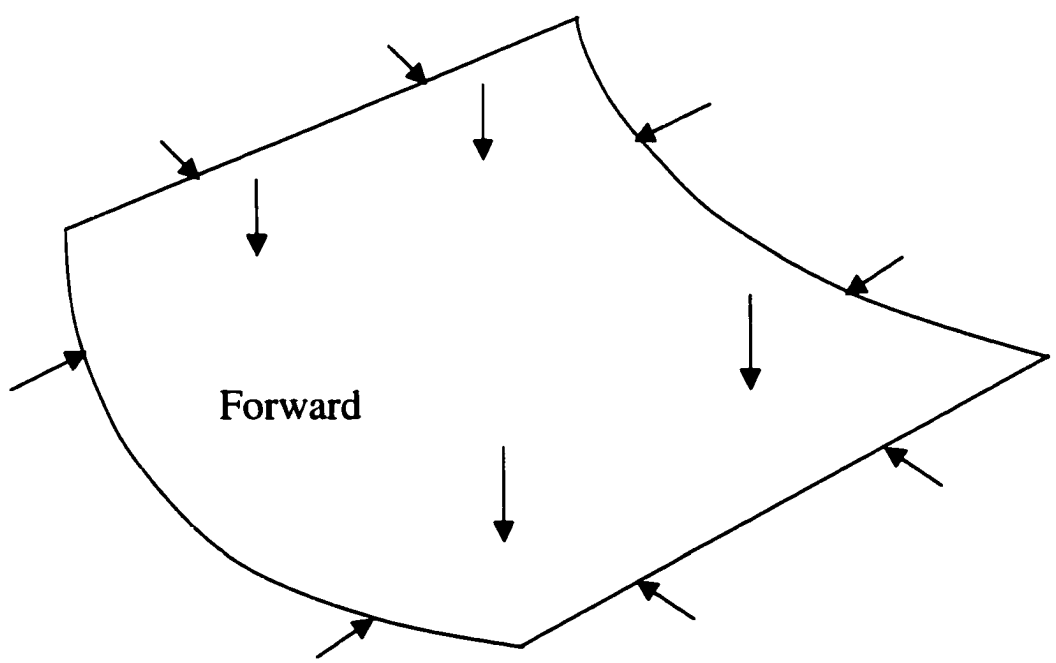

Figure 1.1. A data point is a "12-dimensional" vector

For each fixture, ten "12-dimensional" vectors were obtained and the original goal was to compare the 2 fixtures and decide which one gives more consistent/better placement of the car hood. As a first step in statistical modeling and analysis for this type of problem, consider a 2-dimensional idealization ignoring the "Up-Down" measurements and replacing the complicated real hood geometry with a rectangle of approximately the same overall dimensions. Then, the data become ten "8-dimensional" vectors for both fixtures A and B (see Figure 1.2).

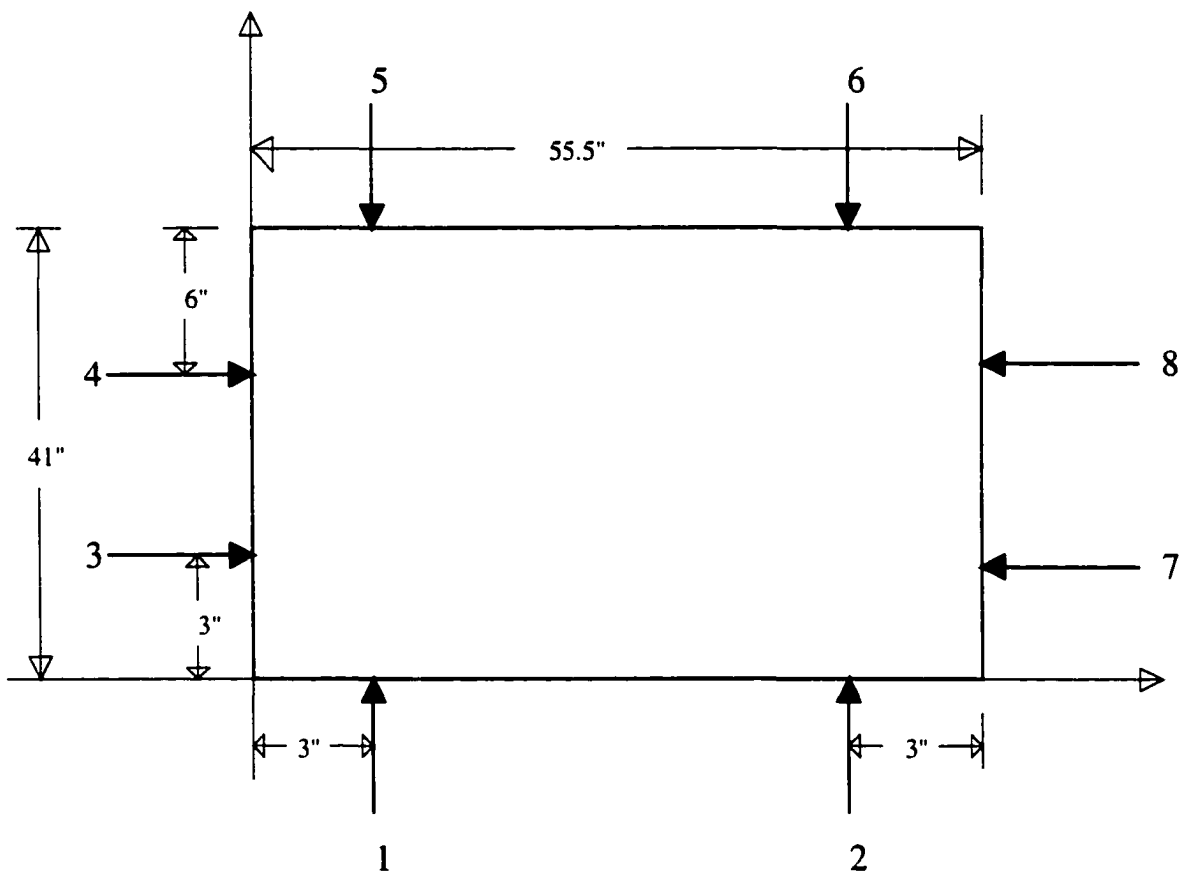

Figure 1.2. (2-D Idealization) A data point is an "8- dimensional" vector

On Figure 1.2, the arrows numbered 1,...,8 indicate the idealized paths taken by the CMM probe. In the idealized car (or CMM) coordinate system, a perfectly placed hood has $\underline{Y} = (Y_1, \ldots, Y_8)^T = (0, 0, 0, 0, 41, 41, 55.5, 55.5)$ where Y_i is the horizontal or vertical coordinate where the hood is touched (i.e., 0 horizontal, 0 horizontal, 0 horizontal, 0 horizontal, 41 vertical, 41 vertical, 55.5 horizontal, and 55.5 horizontal).

A "rough" analysis done by the car-manufacturer analyst was to take ten "8-dimensional" vectors for both fixtures A and B (i.e., $n_A = 10$ vectors $\underline{Y}$ and $n_B = 10$ vectors $\underline{Y}$) and make 8 comparisons of sample variances (one probe path at a time). This ignores geometry effects/physical dependence and answers the wrong question (how variable are the measurements versus how variable is hood placement).

In this dissertation, we consider statistical analyses specifically focused on the issue of "Where is the hood?" (as opposed to "What is the distribution of a Y_i ?"). Our main contribution is in the realm of study planning, where the object is to choose a set of probe paths that provide optimal precision for estimating the position of a single "hood" placement. We propose a figure of merit for comparing alternative designs (data collection plans) and compare several algorithms for optimizing this criterion. Our comparisons of algorithms are across design sizes, "hood" geometries and nominal locations.

CHAPTER 2. LITERATURE REVIEW

There is a fairly small existing statistical literature concerned with the statistical analysis of Coordinate Measuring Machine (CMM) data. To our knowledge, nothing has been done specifically on this problem of study planning for locating known geometries. What follows here is a short review of the published work closest in spirit to ours.

Dowling, Griffin, Tsui and Zhou (1997) described current practices for assessing geometric feature conformance to engineering tolerances using CMM data. They mentioned that the choices of sample design and method of data analysis are fundamentally statistical problems. Current statistical design methods and techniques for estimating surface (of a simple geometric feature) errors are very limited and require restrictive assumptions. In the article, examples involving simple features of 2-dimensional geometries have been used. Although there are difficulties inspecting even simple shapes, there is also an urgent need for practical methodologies for complex forms.

Hulting (1997) commented on the Dowling, et al. article and presented a new way of setting up the problem of fitting geometric models to CMM data that offers advantages over traditional orthogonal distance fitting.

Wang, Gupta, Hulting and Fussell (1998) (in an article derived from Gupta's dissertation) illustrated manufactured part modeling and some statistical methods. An approach to characterizing the geometric variations in aluminum automotive spaceframe extrusions was presented. In such problems, they argued that it may not be possible to recover the full set of shape parameters because of measurement and computation

limitations. So, a regression procedure was presented to select the most important parameters.

Hulting (1995) gave an overview of manufactured part modeling work done at Alcoa. He and his colleagues found that the traditional 3-2-1 method of building a reference frame lacks robustness because a minimum number of points is used and slight errors in any of the six points can greatly affect the resulting frame. Further, while the part localization or "best-fitting" problem has been traditionally viewed as only an optimization problem, he observed that it is really model-fitting and should be considered from a statistical viewpoint. That is, confidence intervals for parameters and/or inferences about form errors (i.e., deviations of an actual manufactured part from design intent) should be obtained. He also mentioned that continued collaboration between the disciplines of engineering and statistics is necessary for statistical thinking to become part of the process of collecting and interpreting coordinate measurement data.

Hulting (1992), looked at doing Gauge R&R studies with multivariate data, in particular, with CMM data. In order to ensure the quality of manufactured parts, the location of features (e.g. holes, slots, surface points) on parts (e.g. subassemblies of automobiles or planes, electronic packages, etc.) are checked for conformance to specifications during manufacturing. These checks involve comparing 3-dimensional (x,y,z) measured coordinates to nominal (design) values. The successful control of the manufacturing process depends, in part, on the validity of these measurements. He argued that a method is needed for analyzing multivariate measurement data to characterize components of variation in observed measurements. Coordinate measurement systems are commonly used in industry, and there has been little work concerning the analysis of data arising from these devices. In his paper,

a multivariate extension of the traditional univariate method for assessing measurement variation is proposed. The method provides a more complete characterization of measurement system performance than the traditional univariate approach.

CHAPTER 3. MODELS AND METHODS

As a first step in developing methods for the statistical planning of CMM studies, we limit our discussion to a 2-dimensional idealization. The design considerations here are for estimation in the simple problem of locating a fixed, known geometry. We investigate how shape and positioning affect optimal design of data collection (which and how many probe paths to employ) and the performance of algorithms intended to identify the optimal designs. This chapter presents the basic statistical modeling and inference we propose for part location, the scope of this dissertation, a design figure of merit, and some computational methods for optimization of the design.

3.1. STATISTICAL MODELING

In this dissertation, we restrict our attention to data collected using probe paths parallel to CMM coordinate axes. And henceforth, for a fixed ideal 2-dimensional object (with its own coordinate system located at the geometry's center of mass), use the following notation:

u	x (horizontal) translation (from the ideal) of the origin of the object's coordinate system,
v	y (vertical) translation (from the ideal) of the origin of the object's coordinate system,
θ	angle of rotation (from the ideal) of the object's coordinate system,
$\underline{\beta} = (u, v, \theta)^T$	fixed but unknown parameter vector describing actual location,

k	number of CMM measurements to be made,
$\underline{Y}$	a k-dimensional response vector (data taken from the CMM) indicating the points at which the probe first touches the object,
$\underline{X}$	a corresponding explanatory variable indicating the k paths taken by the probe (indicates direction of approach, i.e. from top, from bottom, from left or from right and the "level" of that approach),
$\underline{f}(\cdot)$	a (vector of) shape dependent functions,
m	total number of (fixed) candidate probe-paths,
σ^2	repeatability error variance,
$\underline{\varepsilon}$	vector of measurement errors; $\underline{\varepsilon} \sim N_k(\underline{0}, \sigma^2 \underline{I}_k)$.

Supposing the object to be a rigid body, with no measurement error,

$$\underline{Y} = \underline{f}(\underline{X}; \underline{\beta})$$

for an appropriate $\underline{f}$. To get a statistical version of this model, measurement errors are added and the model becomes

$$\underline{Y} = \underline{f}(\underline{X}; \underline{\beta}) + \underline{\varepsilon} \tag{3.1}$$

where $\underline{Y}$, $\underline{f}$, and $\underline{\varepsilon}$ are $k \times 1$, $\underline{X}$ is $k \times 2$ and $\underline{\beta}$ is a 3×1 fixed but unknown parameter vector. This is the model used in our analysis. We note however, that potentially important generalizations of (3.1) exist. For example, one might incorporate uncertainty/variability in realized probe path by letting $\underline{\delta}$ be a vector of k random deviations from the nominal/intended "levels" of approach indicated in $\underline{X}$ and obtain a model

$$\underline{Y} = \underline{f}(\underline{X}, \underline{\delta}; \underline{\beta}) + \underline{\varepsilon}$$

for an appropriate $\underline{f}$.

3.2 STATISTICAL INFERENCE

Model (3.1) is in the form of a nonlinear regression model. Under the distributional assumptions already imposed on $\underline{\varepsilon}$ and the assumption that $\underline{f}$ is smooth, standard statistical theory gives an approximate distribution for the least squares estimator of the parameter vector $\underline{\beta} = (u, v, \theta)^T$. See Seber and Wild (1989, Chapter 2) for details.

For purposes of this dissertation, we need the approximate variance-covariance matrix of $\hat{\underline{\beta}}$, the least squares estimator of $\underline{\beta}$. This depends on the "design" (i.e., k and the probe paths indicated in $\underline{X}$). The approximate distribution of $\hat{\underline{\beta}}$ is MVN $(\underline{\beta}, \sigma^2 C^{-1})$ where $C = F^T F$, for F an appropriate k×3 matrix of derivatives. That is, think of a single "X" as X = (S, L) where S is "Side" and L is the "Level". For example, X = (2, 3) might mean "Side 2" = "left" and "Level 3" = "vertical coordinate = 3." Then $f(X, \underline{\beta})$ is the horizontal or vertical coordinate at which a probe following path X first touches the object located as $\underline{\beta}$. And then

$$F(\underline{\beta}) = \begin{bmatrix} \frac{\partial f(X_1, \underline{\beta})}{\partial u} & \frac{\partial f(X_1, \underline{\beta})}{\partial v} & \frac{\partial f(X_1, \underline{\beta})}{\partial \theta} \\ \vdots & \vdots & \vdots \\ \frac{\partial f(X_k, \underline{\beta})}{\partial u} & \frac{\partial f(X_k, \underline{\beta})}{\partial v} & \frac{\partial f(X_k, \underline{\beta})}{\partial \theta} \end{bmatrix} \tag{3.2}$$

and one wants "small" matrices $C^{-1} = (F^T F)^{-1}$. We proceed to propose a figure of merit based on C^{-1}.

3.3. DESIGN FIGURE OF MERIT

The objective here is to identify "good" designs. To create a doable problem, we limit consideration to designs where each "X" belongs to some pre-specified finite set Δ. Δ might consist of (m/4) "equally spaced" paths into each side of a rectangle bounding the object in nominal position as shown in Figure 3.1.

For a choice of k "X"s, $\underline{X} = (X_1, X_2, \ldots, X_k)$ is a data collection recipe, a design. For a fixed $\underline{\beta}$, computation of C^{-1} is straightforward. But optimization of a function of C^{-1} for a single $\underline{\beta}$ begs the question that one doesn't know $\underline{\beta}$ at the data collection design stage.

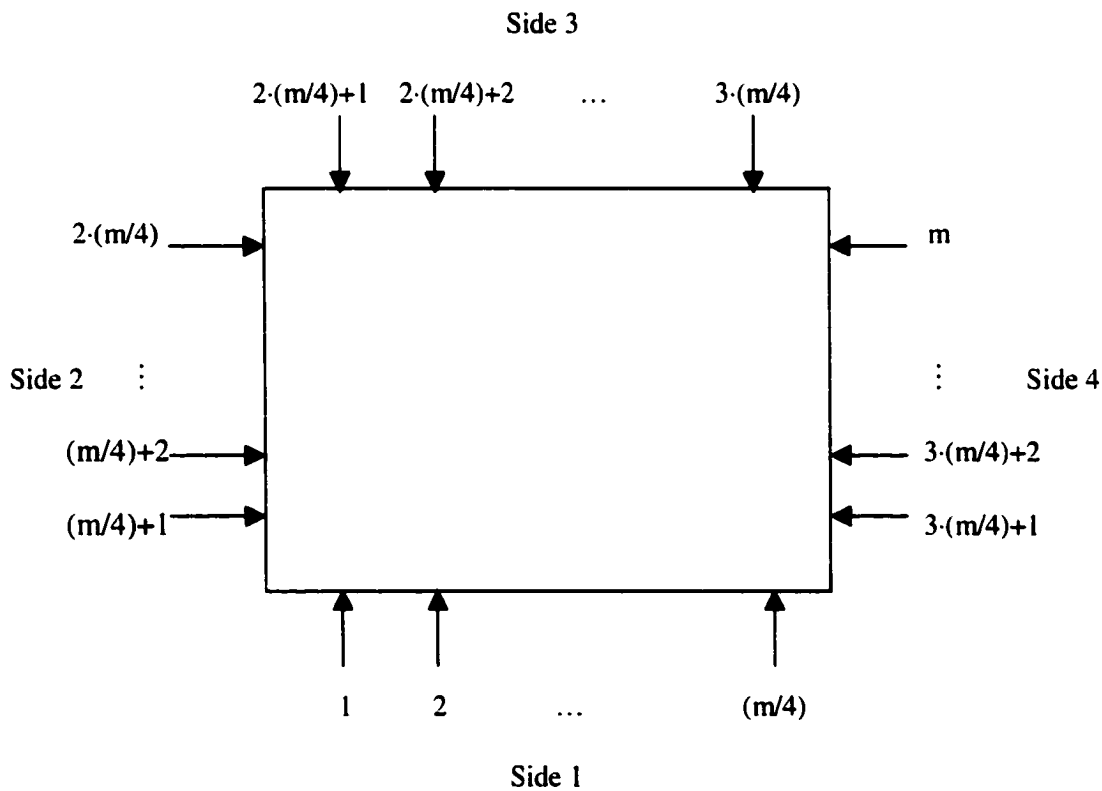

Figure 3.1. A pre-specified finite set Δ with (m/4) paths from each of 4 sides (The rectangle bounds the object in $\underline{\beta} = \underline{0}$ or nominal position)

Suppose, however, that one can bound the entries of the vector $\underline{\beta}$, as

$$u_L = -u_H \le u \le u_H,$$

$$v_L = -v_H \le v \le v_H,$$

$$\theta_L = -\theta_H \le \theta \le \theta_H. \tag{3.3}$$

(These are a priori constraints on the actual position of the object.) Then considering "high-medium-low" combinations of these positioning parameters, we can define 27 different vectors $\underline{\beta} = (u, v, \theta)^T$ and corresponding object positions. (We emphasize that the potential probe paths in Figure 3.1 are defined in terms of the part coordinate system when the part is in its nominal ($\underline{\beta} = \underline{0}$) position and are therefore fixed with respect to the CMM coordinate system.) We then propose as a figure of merit for a design $\underline{X}$,

$$\text{Ave}(WT(\underline{X})) \tag{3.4}$$

for $$WT(\underline{X}) = \omega_1 d_1(\underline{X}) + \omega_2 d_2(\underline{X}) + \omega_3 d_3(\underline{X})$$

where Ave is an average over 27 vectors $\underline{\beta}$ as defined by (3.3); and d_1, d_2, d_3 are the 3 diagonal elements of C^{-1}; and ω_1, ω_2, and ω_3 are non-negative weights. In particular, we reason that the choice $\omega_1 = \omega_2 = 1$ and $\omega_3 = r^2$, for r the largest distance from the center of mass to a point on the boundary of the geometry of interest, is a sensible one. As the d_i are approximate variances of $\hat{u}$, $\hat{v}$, and $\hat{\theta}$, this choice penalizes equally misperceptions of the location of "extreme points" of a geometry traceable to imprecision in estimation of horizontal translation, vertical translation and rotation of the geometry. This figure of merit is roughly the average weighted sum of the 3 variances: variance of the estimated horizontal translation, variance of the estimated vertical translation, and variance of the estimated

rotation. Referring to the original goal in Chapter 1, the issue of "Where is the hood?" can be addressed using this criterion to compare data collection designs.

3.4. COMPUTATIONAL METHODS

3.4.1. Exhaustive Computation

Before resorting to search heuristics, it is sensible to get an idea how "big" a particular problem is and whether direct enumeration of all possible designs is feasible. We count the number of different designs (for a given m and k combination). For the purpose of illustration, consider a small example with m = 4 and k = 4. The design $X_1 = 1, X_2 = 2, X_3 = 4, X_4 = 2$ is the same as the design $X_1 = 4, X_2 = 1, X_3 = 2, X_4 = 2$ and is also the same as the design $X_1 = 2, X_2 = 2, X_3 = 4, X_4 = 1$, etc. In counting distinct designs, we may therefore simply count the number of different vectors $\underline{X}$ with $X_1 \le X_2 \le \ldots \le X_k$. This, for given m and k is

$$\sum_{i1=1}^{m} \sum_{i2=i1}^{m} \sum_{i3=i2}^{m} \cdots \sum_{ik=ik-1}^{m} 1 = \binom{m+k-1}{k}. \tag{3.5}$$

For m = 20, and k = 4, using the (3.5) one can count 8,855 different designs; for m = 20 and k = 8, again using (3.5) one can count 2,222,075 different designs; etc., other counts are given in Table 3.1. In order to develop an understanding of the complexity of the

Table 3.1 The numbers of designs for various m and k combinations

m	k = 4	k = 8	k = 10	k = 20
20	8,855	2,220,075	20,030,010	68,923,264,410
40	123,410	314,457,495	8,217,822,536	
80	1,837,620	58,433,559,570	5,085,018,206,136	
160	28,342,440	12,655,529,067,060		
1,000	41,917,125,250			
2,000	668,668,500,500			

enumeration required by an exhaustive search over candidate designs, we wrote a program (cnt.c) to simply count designs (not evaluate weighted traces) using the looping as indicated on the left of (3.5). The empty cells in Table 3.1 indicate that even the problem of simply enumerating designs is too big to handle at present. Hence, direct enumeration of values of criterion (3.4) for all designs is feasible only for relatively small problems. Some heuristic algorithms are thus needed to do searches for designs with small values of criterion (3.4) in "big" problems.

For the purpose of providing a basis of comparison (to an exact optimum) for results obtained using our heuristics, we do make exhaustive searches for optimal designs in small problems. Computer programs for the exhaustive searches are in Appendix B.

3.4.2. One-at-a-time Search Heuristic (ONE)

Consider m possible X's (i.e., probe paths), (m/4) corresponding to approach from each one of 4 directions. Further, suppose that we can afford k measurements. What we will call a one-at-a-time search heuristic (ONE) does the following. For a single iteration, make a random choice of $X_1, X_2, \ldots, X_{k-1}$. Optimize over choices of X_k. Then holding $X_1, X_2, \ldots, X_{k-2}, X_k$ fixed, optimize over X_{k-1}. Then holding $X_1, X_2, \ldots, X_{k-3}, X_{k-1}, X_k$ fixed, optimize over X_{k-2}. Continue until finally, holding $X_2, X_3, \ldots, X_k$ fixed, optimize over X_1. Then the final design and the corresponding average weighted trace are recorded.

Typically in practice, several iterations of such an algorithm are made. Here we repeat the above many times to get many sets of final designs and average weighted traces. Based on these many average weighted traces, we build a frequency distribution table to characterize performance of this search heuristic. Its use will be discussed in Sections 3.4.4

and 3.4.5. Computer programs for the One-at-a-time (ONE) search heuristic are in Appendix B.

3.4.3. Genetic Algorithm (GA)

Michalewicz (1996) discusses Genetic Algorithms (GAs) and Evolution Programs (EPs). We will use a discrete (optimization) version of a genetic algorithm. Since our GA here does not involve binary code, it might more properly be called an EP. However, Michalewicz (1996) mentions that it is impossible to draw a clear line between GAs and EPs and for the sake of using a familiar name, we will use the "GA" terminology.

Our GA operates as follows. Initially, the parameters population size (POPSIZE), maximum generations (MAXGENS), size of the design (k), number of possible paths (m), probability of crossover (PXOVER) and probability of mutation (PMUTATION) are chosen. To initialize a population, some POPSIZE number of k-gene chromosomes (k-point data collection designs) are set randomly. This population is the set of initial (potential) solutions.

The algorithm then operates on the existing population. In each generation, each chromosome (design) is evaluated using an evaluation or fitness function which we take to be the reciprocal of the average weighted trace (3.4). Then relative fitness and cumulative fitness (for the designs listed in order of fitness) are also computed. Next, in a selection process, a new population is selected via independent random draws from the probability distribution defined by the fitness values (i.e., proportional to the relative fitness of the elements of the current population). Typically, some chromosomes/designs are selected more than once for inclusion in the next generation. The "best" chromosomes get more copies, the "average" stay even, and the "worst" die off.

Next, a "recombination" operator, crossover, is applied to the individuals/chromosomes in the new population by considering them one at a time in some order. The specifics of our crossover algorithm are these. As we consider a chromosome, we generate a Uniform (0,1) number. If that number is less than PXOVER, the chromosome under consideration is paired with another randomly selected chromosome from the population and a single point crossover is done. (A random integer number, pos, is generated from the uniform distribution on the integers 1 to k-1. The number pos indicates the position of the crossing point. The genes (i.e., probe paths) listed in the chromosomes (designs) to the left of this crossing point are swapped between the pair.) Then consideration is turned to the next chromosome. And so on through the POPSIZE number of chromosomes.

The next operator, mutation, is performed on a gene-by-gene basis across the whole population. Genes (individual probe paths in a design) are selected for mutation independently with fixed probability (PMUTATION). The expected number of mutated genes is then PMUTATE · k · POPSIZE.

Following selection, crossover, and mutation, the new population is ready for evaluation. This evaluation is used to build the probability distribution for the next selection process. After evaluation, there is finally consideration of the elitist function. This compares the best chromosome (design) of current generation to the previous generation's best chromosome. If the best chromosome of the current generation is worse than the best chromosome of the previous generation, the latter replaces the worst chromosome of the current population. Hence, this elitist function upgrades the best individual in each generation. The best chromosome (design) in the final population is the output of the GA

heuristic. The number of generations employed depends on the size of problem and computation resource limitations.

Typically, in practice several iterations of such a GA algorithm are made. Here we have repeated the GA heuristic above many times to produce many final designs and corresponding average weighted traces. Again, a frequency distribution table of final best AWT's is built. The computer programs for the Genetic Algorithm (GA) are in Appendix B.

3.4.4. Comparison of the 2 algorithms (ONE versus GA)

To make fair comparisons of the 2 algorithms, ONE and GA, we need to limit them to approximately the same computation resources. The computing time associated with random number generation in the two algorithms is negligible. For a given design, the steps of calculating the derivatives, computing F^TF and inverting it, averaging across the 27 β combinations to get an Average Weighted Trace (AWT) are all the same for ONE and GA. So essentially, the computational time required by an iteration of either algorithm is simply proportional to the Number of Average Weighted Traces (# AWT) computed. For n_1 iterations of the ONE algorithm and n_g iterations of the GA algorithm, these are

$$\#\,\text{AWT (ONE)} = m \cdot k \cdot n\_1 \tag{3.6}$$

and

$$\#\,\text{AWT (GA)} = (\text{POPSIZE}) \cdot (\text{MAXGENS}) \cdot n\_g. \tag{3.7}$$

Comparable (in terms of computing effort) numbers of runs of ONE and GA are then chosen by setting # AWT (ONE) in (3.6) equal to # AWT (GA) in (3.7). This requires

$$n\_g = \frac{m \cdot k}{(\text{POPSIZE}) \cdot (\text{MAXGENS})} \cdot n\_1. \tag{3.8}$$

That is, one run of ONE is approximately equivalent to $(m \cdot k)/[(\text{POPSIZE}) \cdot (\text{MAXGENS})]$ runs of GA in terms of computational complexity.

3.4.5. Mean "Best" Average Weighted Trace versus Computation Resource

Suppose now that one finds a relative frequency distribution for the final Average Weighted Trace (AWT) from single applications of either ONE or GA. Let W be the final AWT from an additional single run through ONE or GA. The (empirical) relative frequency distribution serves as an approximation of the probability distribution for W. With $w_1 < w_2 < \ldots < w_h$ the distinct final weighted traces seen, we will use the notation

$$p_1 = P(W = w_1),\ p_2 = P(W = w_2),\ \ldots\ ,\ p_h = P(W = w_h) \tag{3.9}$$

for the relative frequencies/probabilities.

Now consider the distribution of the best AWT found in n future runs through ONE or GA. That is, assume $W_1, W_2, \ldots, W_n$ are iid with distribution (3.9) and let Z_n be the minimum of $W_1, W_2, W_3, \ldots, W_n$. Use the notation

$$q_1 = P(Z_n = w_1),\ q_2 = P(Z_n = w_2),\ \ldots\ ,\ q_h = P(Z_n = w_h) \tag{3.10}$$

for the probability mass function of Z_n. The probabilities q_i in (3.10) can be obtained as follows.

$$\begin{aligned} q_i &= P(Z_n = w_i) \\ &= P(Z_n \geq w_i) - P(Z_n \geq w_{i+1}) \end{aligned} \tag{3.11}$$

where,
$$\begin{aligned} P(Z_n \geq w_i) &= P(W_1 \geq w_i,\ W_2 \geq w_i,\ \ldots\ ,\ W_n \geq w_i) \\ &= P(W_1 \geq w_i)^n \\ &= (p_i + p_{i+1} + \ldots + p_h)^n. \end{aligned} \tag{3.12}$$

Rearranging (3.11) using (3.12),

$$q_i = (p_i + p_{i+1} + \ldots + p_h)^n - (p_{i+1} + p_{i+2} + \ldots + p_h)^n. \tag{3.13}$$

Now, a useful summary of the distribution described by (3.13) is

$$E(Z_n) = \sum_{i=1}^{h} w_i \cdot q_i . \tag{3.14}$$

The limit of this expected value in n can be obtained as follows. Using (3.13), it can be seen that

$$\lim_{n\to\infty} q_i = \begin{cases} 1, & \text{if } i = 1 \\ 0, & \text{otherwise} \end{cases}$$

and hence,

$$\lim_{n\to\infty} E(Z_n) = w_1 . \tag{3.15}$$

Using (3.14) as the mean "best" AWT (Average Weighted Trace) found in n runs through either the ONE or GA heuristic, a graph of $E(Z_n)$ versus n can be plotted. To make a "fair" comparison of the methods, $E(Z_{n\_l})$ for the ONE heuristic should be compared with $E(Z_{n\_g})$ for the GA heuristic, where n_g is given by (3.8). (Obviously, (3.15) indicates that up to the precision provided by the simulations conducted to find the distributions (3.9), for very large n_1 and n_g the algorithm with the smaller value of w_1 will be preferred.)

3.5. AN EXAMPLE CASE

Before closing this discussion of methods and proceeding to the results (Chapter 4), we present a more detailed walk-through of an example case. For the sake of demonstration, we will go through a Triangle Shape, 0° nominal orientation case.

Following the procedure in Appendix A.2, we obtained the necessary (shape-dependent) functions whose derivatives are needed to compute the derivative matrix (3.2). In this particular case, there are 6 functions of $\underline{\beta}$: f1, f2, ..., f6 that specify the vertical and horizontal levels at which probe paths first touch the triangle. (As shown in Figure 3.2, "y"

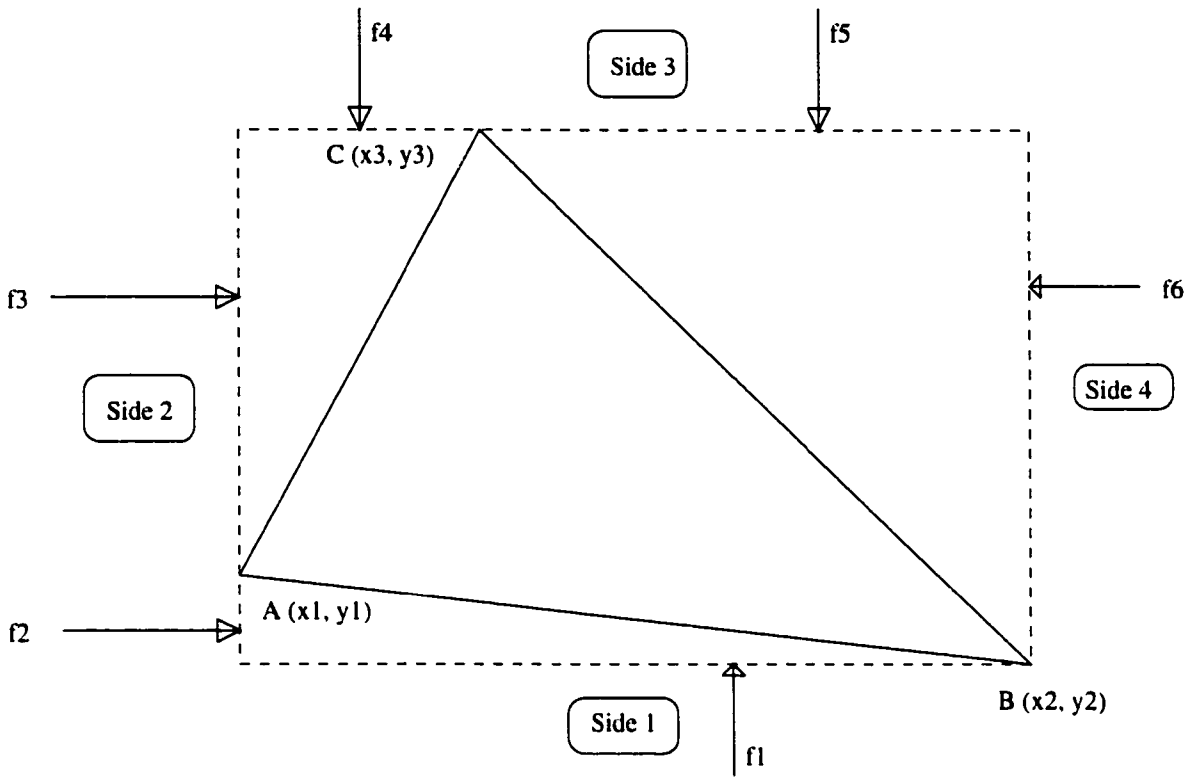

Figure 3.2 Triangle Shape, 0° Orientation, Nominal Position (i.e., $\underline{\beta} = \underline{0}$)

coordinates corresponding to paths from below are specified by f1, "x" coordinates corresponding to paths from the left are specified by f2 and f3, "y" coordinates corresponding to paths from above are specified by f4 and f5, and "x" coordinates corresponding to paths from the right are specified by f6.) A partial derivative symbolized as a partial of "f" in expression (3.2) is then a partial of one of these functions f1, f2, ..., f6 specified by the probe path (direction and level) and value of $\underline{\beta}$. The partial derivatives can be obtained using the program "der.s" in Appendix B.

Consider the "small" (m, k) case, i.e. , (m, k) = (20, 4). Hence, as in Figure 3.1, we have m = 20 (fixed) probe paths, Δ = m/4 = 5 paths for each of the 4 sides. We want to find the best design subject to these conditions, i.e., we want to locate the k = 4 (possibly non-distinct) paths from the m = 20 "possible" probe paths, with the ultimate goal of answering

the question "Where is the Triangle?" By (3.5), there are 8855 (see Table 3.1) possible designs.

To be concrete, take one design out of the 8855 different possibilities consisting of probe paths 6, 1, 11, 20 (i.e., $X_1 = 6$, $X_2 = 1$, $X_3 = 11$, $X_4 = 20$). Using values $u_L = -0.25$ inch, $u_H = 0.25$ inch, $v_L = -0.25$ inch, $v_H = 0.25$ inch, $\theta_L = -0.1$ radian, $\theta_H = 0.1$ radian, for (3.3), we obtain 27 different $\underline{\beta}$'s. For each, we compute

$$F(\underline{\beta}) = \begin{bmatrix} \frac{\partial f2(6,\underline{\beta})}{\partial u} & \frac{\partial f2(6,\underline{\beta})}{\partial v} & \frac{\partial f2(6,\underline{\beta})}{\partial \theta} \\ \frac{\partial f1(1,\underline{\beta})}{\partial u} & \frac{\partial f1(1,\underline{\beta})}{\partial v} & \frac{\partial f1(1,\underline{\beta})}{\partial \theta} \\ \frac{\partial f4(11,\underline{\beta})}{\partial u} & \frac{\partial f4(11,\underline{\beta})}{\partial v} & \frac{\partial f4(11,\underline{\beta})}{\partial \theta} \\ \frac{\partial f6(20,\underline{\beta})}{\partial u} & \frac{\partial f6(20,\underline{\beta})}{\partial v} & \frac{\partial f6(20,\underline{\beta})}{\partial \theta} \end{bmatrix},$$

where by the notation $fj(X, \underline{\beta})$ we are making explicit the fact that f1, f2, ..., f6 are functions not only of $\underline{\beta}$, but also of horizontal or vertical level associated with path X and the fact that none of the 27 $\underline{\beta}$'s is large enough to change the functions fj relevant for probe paths 6, 1, 11, and 20. For each of the 27 $F(\underline{\beta})$'s, $C^{-1} = (F^TF)^{-1}$ is computed, as is the corresponding $WT(\underline{X})$. Next, $Ave(WT(\underline{X})) = 0.532119$ as in (3.4) is obtained.

Using the same procedure as above, we can calculate the AWT, $Ave(WT(\underline{X}))$, for the remaining 8855 - 1 = 8854 designs. The minimum one(s) correspond to the best design(s). Through the Exhaustive Computation (see program output in Appendix C), the best/minimum AWT is found to be $Ave(WT(\underline{X})) = 0.438142$ with the design $X_1 = 6$, $X_2 = 11$, $X_3 = 20$, $X_4 = 20$ as indicated in Figure 3.3.

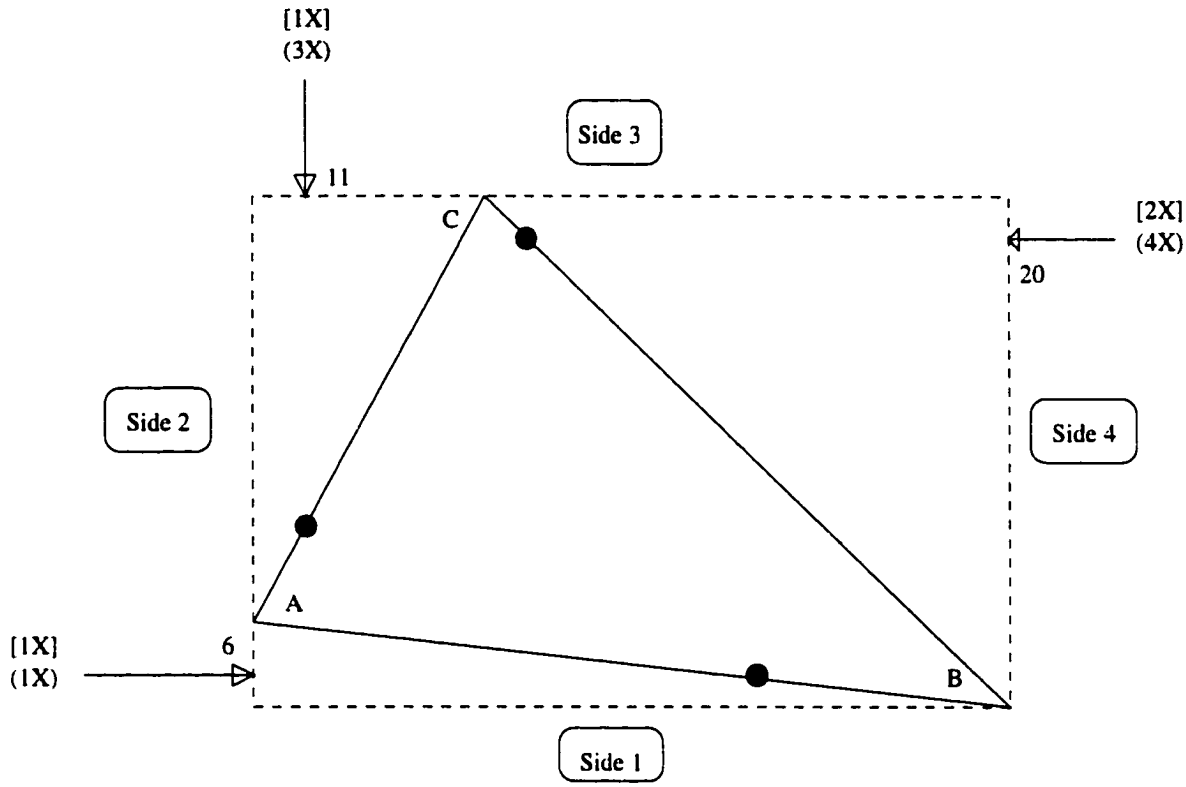

Figure 3.3. Best Small/Medium (m, k) Designs for Triangle Shape, 0° Orientation; Nominal Position (i.e., $\underline{\beta} = \underline{0}$) of Triangle is Shown

In Figure 3.3, the best designs (Triangle shape, 0° nominal orientation) for small and medium (m, k) are displayed. The arrows indicate probe paths and black dots represent the points at which the probe touches the triangle if in nominal position for the k = 4 (and k = 8) trials. For k = 4, "[1X]" indicates that paths 6 and 11 are used once each and "[2X]" indicates that path 20 is used twice. For k = 8, "(1X)", "(3X)", and "(4X)" indicate that paths 6, 11 and 20 are used respectively once, three times and four times.

Figures like Figure 3.3 giving optimal designs for other combinations of geometries/orientations, and small/medium (m, k), are provided in Appendix D.

CHAPTER 4. RESULTS AND DISCUSSION

We have applied the methods presented in Chapter 3 to 4 test geometries ("rectangle," "triangle," "hood shape," and "parabolas"), 2 nominal orientations for the objects ("0°" and "45°"), and 3 sets of m and k (small, medium, large). Figures A.1 through A.8 in Appendix A show the eight combinations of geometry and nominal placements. As shown on Figures 1.2 and 3.1, we have restricted attention to 4 types of CMM probe paths: from Side 1, Side 2, Side 3 and Side 4, on lines parallel to the CMM coordinate axes. In our study, the 3 sets of "small-medium-large" (m, k) are (20, 4), (20, 8), and (40,16). For the large (m, k) combination, we could not afford an exhaustive search for an optimum design and can only apply the ONE and GA algorithms.

Before going on we need to describe more precisely the set of paths indicated in Figure 3.1. After obtaining the horizontal and vertical (x and y) coordinates of the "extreme" points (as prescribed in Appendix A) of a test geometry when $\underline{\beta} = \underline{0}$, the lines forming the smallest "rectangle" that bounds the ideally placed geometry are then obvious. We then allow some "margin" on each end of the line segments forming the rectangle and partition the remainder equally to locate the (m/4) probe paths into each side. If desired, one can print out (as an option in the program) the exact locations of the equally spaced probe paths. The "margin" can be changed in the program "con.h". For the "parabolas" shape, the "margin" was taken to be 13.5 inches. For the other 3 shapes, 0.5 inch was used. Without a "margin" any probe path on the "extremes" of the rectangle will miss the object for some $\underline{\beta} \neq \underline{0}$. Computing-wise, that would cause numerical problems. In practical terms, using that path

has the potential to provide only the very crude information "the object lies to one side of the line of approach" rather than the much more informative "measured location of contact."

Appendix A summarizes an example of the analytic geometry and calculus necessary to support our analysis of the 8 = 4×2 different geometry/orientation combinations. Appendix B contains the C and S-Plus programs written to implement the calculations described in Chapter 3. Examples of the output of our programs (i.e., the best/final AWT(s) and their corresponding design(s)) are given in Appendix C. Appendix D contains figures portraying best designs found by direct enumeration and some examples of what the (ONE and GA) algorithms found in cases where no direct enumeration was possible. Appendix E provides graphical comparisons of the performance of the ONE and GA heuristics in terms of expected final AWT as a function of computing effort. This chapter contains our summary discussion of the results of our experimentation with the search algorithms across the 8 geometry/orientation test cases.

4.1. CHOICE OF CONSTANTS IN THE FIGURE OF MERIT

Displays (3.3) and (3.4) prescribe averaging over some set of 27 $\underline{\beta}$'s. These are meant to in some sense represent or cover the range of possible actual positions of the geometry. In the motivating example, actual hood placement didn't vary drastically from the nominal placement. This suggests choices of u_L, u_H, v_L, v_H, θ_L, and θ_H in (3.3) all "close to 0." In our experiments we used

$$u_L = -0.25 \text{ inch}, u_H = 0.25 \text{ inch},$$

$$v_L = -0.25 \text{ inch}, v_H = 0.25 \text{ inch},$$

$$\theta_L = -0.1 \text{ radian}, \theta_H = 0.1 \text{ radian}. \quad (4.1)$$

We have also used the set of larger possible position perturbations defined by

$$u_L = -5 \text{ inch}, u_H = 5 \text{ inch},$$

$$v_L = -5 \text{ inch}, v_H = 5 \text{ inch},$$

$$\theta_L = -0.1 \text{ radian and } \theta_H = 0.1 \text{ radian}$$

with several combinations of geometries/orientations and (m, k) sets. We obtained results generally comparable to those for the choices (4.1). That is, most of "best" designs for the two sets of constants found via complete enumeration are equivalent. See Table 4.1. In Table 4.1, those cases where a single design is optimal for both sets are marked "=" and those cases where optimal designs are slightly different are marked "≠". Where they are different, we have matched up the maximum number of like X_i's in the two designs and report the number (among k) that are unmatched. Those paths which are different tend to be located in

Table 4.1 Comparison of exhaustive search results for the 2 sets of constants $\{u_L, u_H, v_L, v_H, \theta_L, \theta_H\}$

Geometry/ Orientation	(m = 20, k = 4)	(m = 20, k = 8)
Hood0	=	≠ (3)
Hood45	=	=
Par0	≠ (3)	≠ (1)
Par45	≠ (2)	=
Rec0	=	≠ (1)
Rec45	≠ (1)	=
Tri0	≠ (1)	≠ (2)
Tri45	=	=

different "extreme" locations or result from slightly different patterns of repetition among the same set of paths. The differences in Table 4.1 come from very large uncertainty in horizontal and vertical locations of the geometry center of mass specified by our second set of constants. In spite of this, half of the Table 4.1 results indicate a single "best" design.

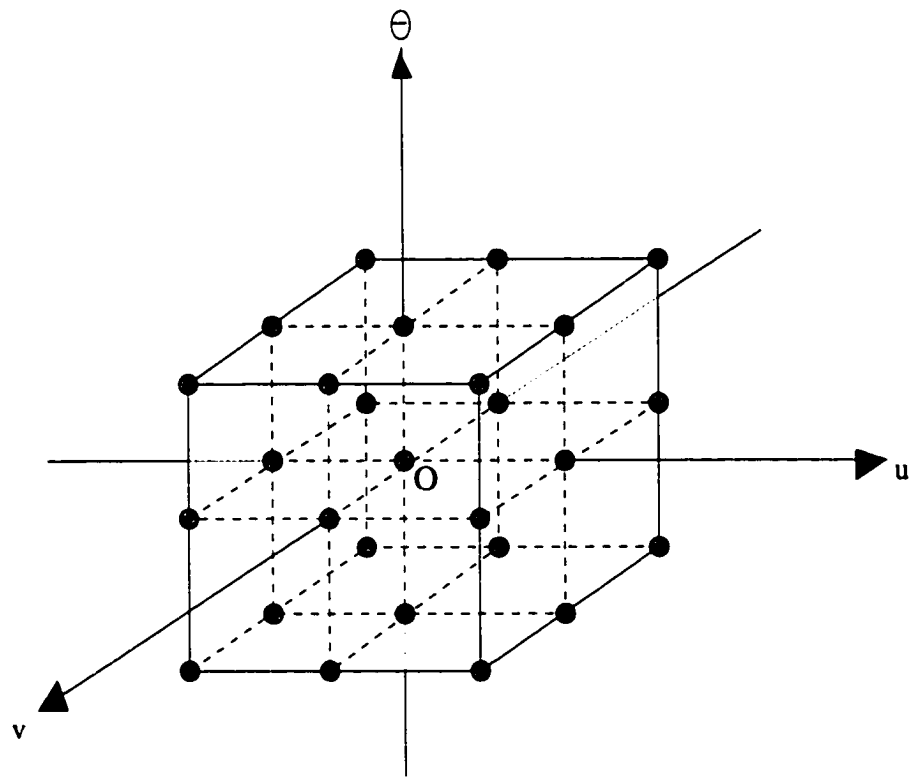

Figure 4.1 27 different $\underline{\beta}$'s

In Figure 4.1, the 27 dots representing the 27 different $\underline{\beta}$'s corresponding to the choice (4.1) are (actually) very close together to the "middle value," O. The 27 dots represent the 27 $\underline{\beta}$'s for which we compute $F(\underline{\beta})$ as in (3.2) for a given design. We can also think the 27 dots on Figure 4.1 as representing the 27 different positions of a geometry as illustrated in Figure 4.2. For each $F(\underline{\beta})$, $C^{-1} = (F^TF)^{-1}$ is obtained and yields a single $WT(\underline{X})$. Since, we have 27

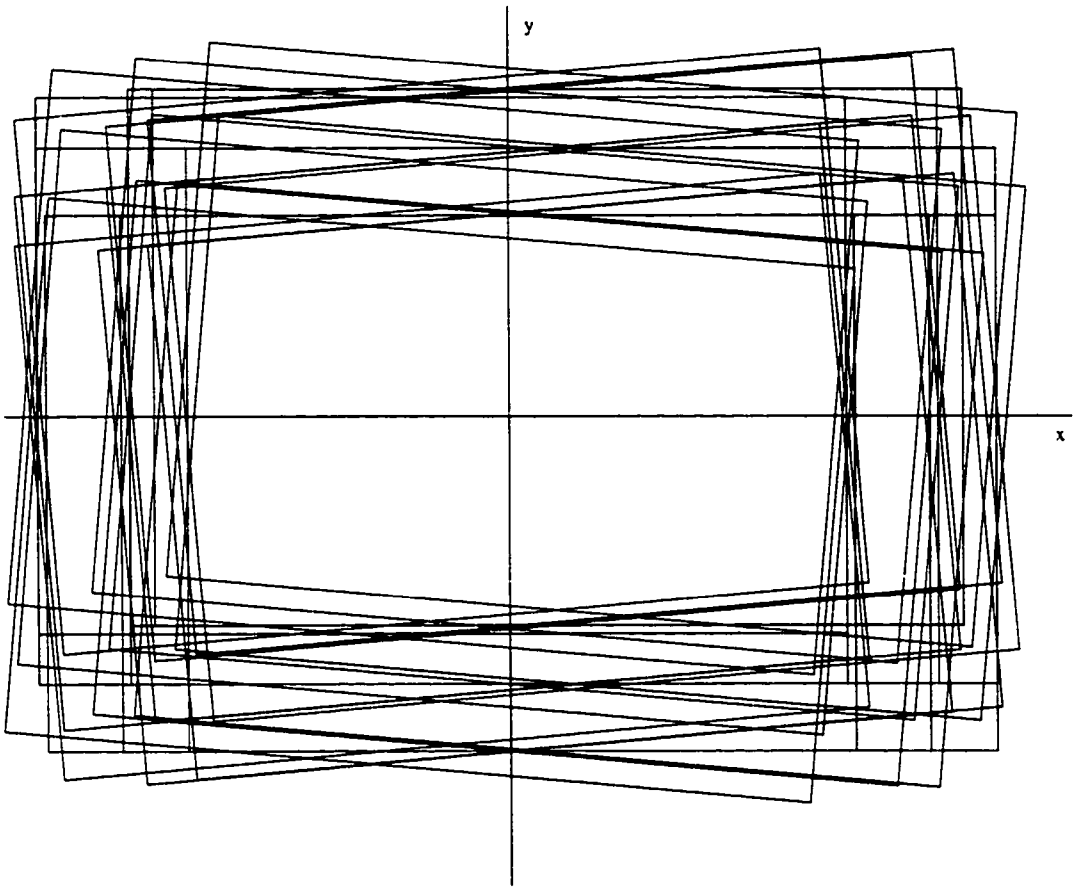

Figure 4.2 27 Different Positions for a Rectangle Corresponding to $\underline{\beta}$'s in Figure 4.1 (Axes are Those for the $u = v = 0, \theta = 0$ Positioning) (Figure Not to Scale)

weighted traces, WT($\underline{X}$), we take their average and the AWT (3.4) is obtained. In the WT calculation, 1 unit of uncertainty in horizontal or vertical position for a geometry's center of mass is penalized roughly the same amount as a unit of uncertainty in location of an "extreme corner" of the object brought on by rotation about its center of mass.

Table 4.2 displays parameters (used in Appendix A) to specify the 4 geometries used as test cases in our study. These were chosen to give test geometries roughly the same "size" as the actual hood in the motivating application.

Table 4.2 Parameters of study geometries (see Appendix A)

Hood	Par	Rec	Tri
r1 = 40"	z = 45"	l_rec = 55.5"	A (-15", -5")
r2 = 90"		w_rec = 41"	B (30", -10")
T1 = 40°			C (-1", 20")
T2 = 20°			

Table 4.3 summarizes the numbers of trials runs made in this study to produce the (empirical) relative frequency distributions described in Chapter 3. In Table 4.3, "Others" refers to "Parabolas/Rectangle/Triangle" shapes, at both 0° and 45°. "Hood" refers to "Hood" shape, at both 0° and 45°.

Table 4.3 Numbers of trials run

Others				Hood			
m	k	GA	ONE	m	k	GA	ONE
40	16	2500	1500	40	16	2500	1000
20	8	2500	5000	20	8	2500	2500
20	4	5000	10000	20	4	2500	5000

Table 4.4 summarizes the GA parameters (POPSIZE AND MAXGENS) used to develop the (empirical) relative frequency distributions for a given geometry/orientation combination. To make a "fair" comparison of the mean "best" AWTs, the (n_g:n_1) ratio in (3.8) can be used. For example, "1:2" (n_g:n_1) ratio means 1 random start of GA algorithm is equivalent in computational burden to 2 random starts of the ONE algorithm.

Table 4.5 summarizes the best/final AWT's obtained through exhaustive search, and using the ONE and GA algorithms for various combinations of geometry/orientation and problem size (m, k). In Table 4.5, the number of trials used for ONE and GA are as shown in Table 4.3. Some figures portraying best/final designs corresponding to the best/final AWTs

Table 4.4 GA parameters, and (n_g:n_l) ratio

Geometry/ Orientation	(m,k)					
	(20,4)		(20,8)		(40,16)	
	POPSIZE/ MAXGENS	n_g:n_l	POPSIZE/ MAXGENS	n_g:n_l	POPSIZE/ MAXGENS	n_g:n_l
Hood0	8/20	1:2	6/40	2:3	8/40	2:1
Hood45	8/20	1:2	8/20	1:1	8/40	2:1
Par0	8/20	1:2	32/250	1:50	20/32	1:1
Par45	8/20	1:2	40/50	2:25	16/40	1:1
Rec0	8/20	1:2	50/64	1:20	32/40	1:2
Rec45	8/20	1:2	40/50	2:25	32/40	1:2
Tri0	8/20	1:2	32/50	1:10	32/80	1:4
Tri45	8/20	1:2	16/40	1:4	16/40	1:1

Table 4.5. Best/final AWT found by enumeration, ONE, and GA

Geometry/ Orientation	Best AWT							
	(m = 20, k = 4)			(m = 20, k = 8)			(m = 40, k = 16)	
	Exhaust	ONE	GA	Exhaust	ONE	GA	ONE	GA
Hood0	0.341623	0.341623	0.341623	0.108169	0.108169	0.108169	0.047245	0.050616
Hood45	0.545409	0.545409	0.545409	0.255715	0.255715	0.255715	0.124980	0.144551
Par0	1.172487	1.172487	1.172487	0.510017	0.510017	0.510017	0.250311	0.256739
Par45	0.457386	0.457386	0.457386	0.211846	0.211846	0.211846	0.105308	0.118118
Rec0	1.507293	1.507293	1.507293	0.752810	0.752810	0.752810	0.376405	0.400188
Rec45	0.755208	0.755208	0.755208	0.370539	0.370539	0.370539	0.185270	0.190907
Tri0	0.438142	0.438142	0.438142	0.193954	0.193954	0.193954	0.060495	0.070156
Tri45	0.050006	0.050006	0.050006	0.020805	0.020805	0.020805	0.009642	0.013417

in Table 4.5 are displayed in Appendix D. Our general discussion of the results are in the following sections.

4.2. EXHAUSTIVE COMPUTATION

For all 4 geometries and 2 orientations in this study, we found the best (minimum) AWT through exhaustive computation for the "small" and "medium" (m, k) (namely (20, 4) and (20, 8)). Corresponding pictures of best designs are in Appendix D. We observe that in some cases because of symmetries in our geometries, placements and choices (4.1), there

exist (and can be many) equally good designs (choices of k and $X_1, X_2, \ldots, X_k$). (We also note, however, that where multiple optima were found in our study, the designs were symmetric. In no case did we find best designs equivalent in terms of AWT that were essentially different.)

The design(s) corresponding to the best (minimum) AWT for the small set (m, k) = (20,4) do not have many paths repeated and the paths prescribed are generally located closest to the "extremes" of the region of study and aimed near nominal "corners" of a particular geometry. This is perhaps not surprising, since those locations are often "furthest" from the center of mass of the geometry (and are thus very important in determining θ, the angle of rotation of the object about its center of mass) and also somehow "capture the shape" of the object.

For the "medium" size problem with (m, k) = (20, 8), optimum probe paths are like those for "small" (m, k). The additional data points generally correspond to probe paths aimed at additional "extremes" or "corners," and some paths are repeated as k grows.

The evidence provided by the exhaustive searches indicates that our figure of merit is reasonable.

4.3. SEARCH HEURISTICS (ONE AND GA)

The particular versions of the GA algorithm used in our study had various POPSIZEs (depending on the geometry/orientation and problem size (m, k)) as in Table 4.4. Michalewicz (1996) notes that if the population is too small, a GA will converge too quickly. On the other hand, if it is too large, the GA may waste computational resources: the waiting time for an improvement can be too long. We determined by trial and error that POPSIZE

and MAXGENS as in Table 4.4 work well in our design problems. Other GA parameters in our study were PXOVER = 0.8 and PMUTATION = 0.15. These were also chosen based on some trial and error experience with the heuristic in this context. These choices provide for frequent cross-overs and relatively infrequent mutations.

As shown in Table 4.5, both ONE and GA found an optimal design for all combinations of geometries/orientations, in the "small" (m, k) = (20, 4) and "medium" (m, k) = (20, 8) problems in the large numbers of trials indicated in Table 4.3.

For the "large" problem with (m, k) = (40, 16), direct enumeration was not possible and we therefore do not know the exactly optimal value of AWT. As indicated in Table 4.5, in all cases, ONE found better designs than GA. However, the differences in the best AWT(s) found are very small and the corresponding designs are qualitatively similar.

The best design(s) corresponding to the best AWT found by ONE and GA for the "large" (m, k) problem have many repeated paths and the paths prescribed are again generally located near the "extremes" of the region of possible location and "corners" of a geometry. The optimum paths are as those in "small" or "medium" (m, k) problems and the additional data points are generally obtained as repeats of the paths prescribed in the "small" or "medium" context.

It seems that as k increases, until "optimal paths" have been selected many times (and with about the same relative frequency), "new" paths do not appear in an optimal design. However, apparently as m increases (improving the "fineness of the grid" of possible probe paths), the paths used in an optimal design are qualitatively the same as for smaller m. Therefore, in some sense, it is k and not m that is the principal determiner of the nature of optimal designs.

Graphs of the Mean "Best" AWT, EZ_n, versus the Computation Resource (n) for all combinations of shapes/orientations, (m, k) for both ONE and GA are plotted in Appendix E. On the graphs, the "Computation Resource (n)" is n_g. What is plotted is then EZ_n for the GA and the corresponding value "EZ_n" for ONE based on trials as in display (3.8) using the parameters (POPSIZE and MAXGENS) in Table 4.4. For all the geometry/orientation combinations studied and "small" (m, k) problems, in at least n_g = 4 (or n_1 = 16) trials, we have an expected best AWT that is essentially the optimum value. For "medium" (m, k), in n_g = 4 (or n_1 = 100) trials, an expected best AWT is essentially the optimum value. Finally, for "large" (m, k) problem, in n_g = 10 (or n_1 = 4) trials, an expected best/final AWT nearly as good as that discovered in our (much larger) study is available.

CHAPTER 5. CONCLUSIONS

In this chapter, a summary for this dissertation will be presented, followed by some recommendations and finally, some possible topics for future research.

5.1. SUMMARY

Using the Figure of Merit in (3.4), several computing methods (Exhaustive, ONE, and GA) have been used to search for optimal designs for various geometries, orientations, design sizes (k) and numbers of candidate probe paths (m).

The summary of the results generally goes as follows. Various geometries, orientations, number of data points (k) and numbers of candidate paths (m) affect which paths enter optimum designs and the corresponding Average Weighted Traces (AWTs). m does not really affect the nature of an optimum design. But the larger m, the higher is the computation time. The number k affects the qualitative nature of optimum designs. Apparently, as k exceeds the number of "extremes" (peaks/valleys/corners) of a geometry (or as there are enough data to "capture" the shape of the object), the paths chosen in an optimal design will just repeat the optimum locations of smaller designs. For the same object shape, different orientations will give different optimum designs and AWTs but the qualitative nature of our conclusions is unchanged. The paths in an "optimum/best" design will generally be located closest to the "extremes" of the object boundary or will be aimed near "corners" of the object. Across our entire study we found no cases where two fundamentally different designs had optimum AWTs. (Only symmetries in a problem produced multiple

optima.) Across our study, the ONE algorithm seems better than the GA algorithm since it almost always gives a smaller Mean "Best" AWT for a given investment of computing effort. However, the differences (see Table 4.5), are very small.

Based on this summary and referring back to the motivating problem in Chapter 1, we see that progress has been made. We have found a useful mathematical formulation of the problem of estimating the unknown location of a known 2-dimensional object using CMM-type data in the presence of measurement error. We have shown that the formulation provides guidance in the planning of CMM-type data collection. And we have taken steps to provide algorithms for optimizing that data collection.

5.2. RECOMMENDATIONS

For a "small" choice of (m, k), one can use an exhaustive search to find a design. It will give an optimum/best solution in a (relatively) "short" time. For a "medium/large" choice of (m, k), the author recommends the ONE algorithm since it will typically give "optimum/best" results in a reasonably "not-too-long" time. The GA algorithm sometimes finds "optimum/best" solutions but generally does not, for "large" (m, k) problem.

5.3. FUTURE RESEARCH

There are several directions in which one can expand this research. First, one might extend it from known 2-dimensional geometries to known 3-dimensional geometries. Second, as we noted in Section 3.1, uncertainty in probe paths might be incorporated into our current model (3.1). Third, instead of employing tedious algebra to produce shape-dependent equations on a geometry-by-geometry basis as outlined in Appendix A, a general

methodology might be developed for linking Computer Aided Design (CAD) files to discrete numerical analysis routines to automate calculations for arbitrary geometries.

APPENDIX A. FORMULAS

This appendix provides more details for the analysis outlined in Chapter 3, for specific/particular geometries/orientations. The prescriptions are presented in Section A.2 for one particular geometry/orientation (Triangle/0° orientation). For other geometries/orientations, the procedures described in Section A.2 apply, and thus in Sections A.3 - A.9, only the figures and our choice of parameters describing them are presented.

A.1. TRANSLATIONS AND ROTATION

From simple analytical geometry, the relationship between a point's coordinates (x", y") in a translated and rotated coordinate system and its coordinates (x, y) in an original coordinate system is

$$(x \quad y) = (u \quad v) + (x'' \quad y'') \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix},$$

which is equivalent to

$$\begin{aligned} x &= u + x'' \cos\theta - y'' \sin\theta \\ y &= v + x'' \sin\theta + y'' \cos\theta \end{aligned} \tag{A.1}$$

where u is the horizontal translation of the origin, v is the vertical translation of the origin and θ is the angle of rotation of the coordinate axes.

A.2. TRIANGLE, 0° ORIENTATION

The following development is used to find the functions to be entered into the program "der.h" for obtaining the derivative matrix F, for Triangle Shape, 0° orientation. The parameters used to describe the triangle in its nominal position (to be entered to the program "con.h") consists of the 3 corner points A(x1, y1), B(x2, y2), C(x3, y3) expressed in an arbitrary/convenient coordinate system with axes parallel to those of the CMM. The center of mass $(\bar{x}, \bar{y})$ with respect to this arbitrary coordinate system is

$$\bar{x} = \frac{x1 + x2 + x3}{3} \quad \text{and} \quad \bar{y} = \frac{y1 + y2 + y3}{3}.$$

For the remainder of our discussion, after we know the center of mass of our geometry (in this case, the triangle), we use this as our origin for a coordinate system on the object, and construct x" and y" axes through this point, parallel to the CMM coordinate axes when the geometry is in its ideal position. Note that if we know from the beginning the center of mass of our object (for example, in the rectangle case), we can omit the above discussion, and begin calculations with the object's center of mass as the origin of our coordinate system.

The following 3 steps can then be taken. First, the coordinates of the 3 extreme/corner points A, B, C can be found. Second, equations of the 3 lines (AB, AC, BC) are derived. Third, the 6 (shape dependent) functions f1, ... , f6 indicated in Figure A.1 and specifying where a probe path first touches the figure are identified.

A.2.1. Step 1: Finding the Extreme/Corner Points A, B, C

The coordinates of A(xA, yA), B(xB, yB) and C(xC, yC) in the object's coordinate system are

$$xA = x1 - \bar{x},$$
$$yA = y1 - \bar{y},$$
$$xB = x2 - \bar{x},$$
$$yB = y2 - \bar{y},$$
$$xC = x3 - \bar{x},$$
$$yC = y3 - \bar{y}.$$

These extreme/corner points are useful in identifying the rectangle that bounds the geometry (in this case, the triangle) when it is in its nominal position. Allowing some "margin" on each end of each side of the bounding rectangle, (m/4) equally spaced candidate probe paths can be identified.

A.2.2. Step 2: Finding the Equations for the 3 Sides of the Triangle Shape

There are 3 equations (in the object's coordinate system) to find: those for lines AB, AC, and BC. From

$$\frac{yB - yA}{xB - xA} = \frac{y - yA}{x - xA},$$

the equation for line AB is

$$y = \frac{y2 - y1}{x2 - x1} x + \frac{(\bar{x} - x1)(y2 - y1)}{x2 - x1} + y1 - \bar{y}$$

which we will write in the triangle's coordinate system as

$$y'' = (p1)\, x'' + (q1) \qquad (A.2)$$

where, $$p1 = \frac{y2 - y1}{x2 - x1} \quad \text{and} \quad q1 = \frac{(\bar{x} - x1)(y2 - y1)}{x2 - x1} + y1 - \bar{y}.$$

The equation for line AC is

$$y = \frac{y3 - y1}{x3 - x1} x + \frac{(\bar{x} - x1)(y3 - y1)}{x3 - x1} + y1 - \bar{y}$$

which we will write in the triangle's coordinate system as

$$y'' = (p2)\, x'' + (q2) \tag{A.3}$$

where, $p2 = \dfrac{y3 - y1}{x3 - x1}$ and $q2 = \dfrac{(\bar{x} - x1)(y3 - y1)}{x3 - x1} + y1 - \bar{y}$.

Finally, the equation of line BC is

$$y = \frac{y3 - y2}{x3 - x2} x + \frac{(\bar{x} - x2)(y3 - y2)}{x3 - x2} + y2 - \bar{y}$$

which we will write in the triangle's coordinate system as

$$y'' = (p3)\, x'' + (q3) \tag{A.4}$$

where, $p3 = \dfrac{y3 - y2}{x3 - x2}$ and $q3 = \dfrac{(\bar{x} - x2)(y3 - y2)}{x3 - x2} + y2 - \bar{y}$.

A.2.3. Step 3: Finding the 6 shape dependent functions

Suppose that (as pictured in Figure A.1) one knows the height (y) or the depth (x) (in a coordinate system with origin at the ideal location for the center of mass with axes parallel to the CMM coordinate axes, i.e., the object's coordinate system when $\underline{\beta} = \underline{0}$) for the horizontal/vertical coordinate at which a probe following path X, from side 1 or side 2 of the bounding rectangle, first touches the side AB of the triangle (actually located at $\underline{\beta}$). Given that y (or x), one can solve the equations (A.1) and (A.2) for x (or y) in terms of the parameter $\underline{\beta} = (u, v, \theta)^T$ and y (or x). If y is known,

$$x = \frac{(y - v - q1 \cdot \cos\theta)(\cos\theta - p1 \cdot \sin\theta)}{(\sin\theta - p1 \cdot \cos\theta)} + u - q1 \cdot \sin\theta = f2(y, u, v, \theta). \tag{A.5}$$

On the other hand, if x is known,

$$y = \frac{(x - u - q1 \cdot \sin\theta)(\sin\theta - p1 \cdot \cos\theta)}{(\cos\theta - p1 \cdot \sin\theta)} + v + q1 \cdot \cos\theta = f1(x, u, v, \theta). \tag{A.6}$$

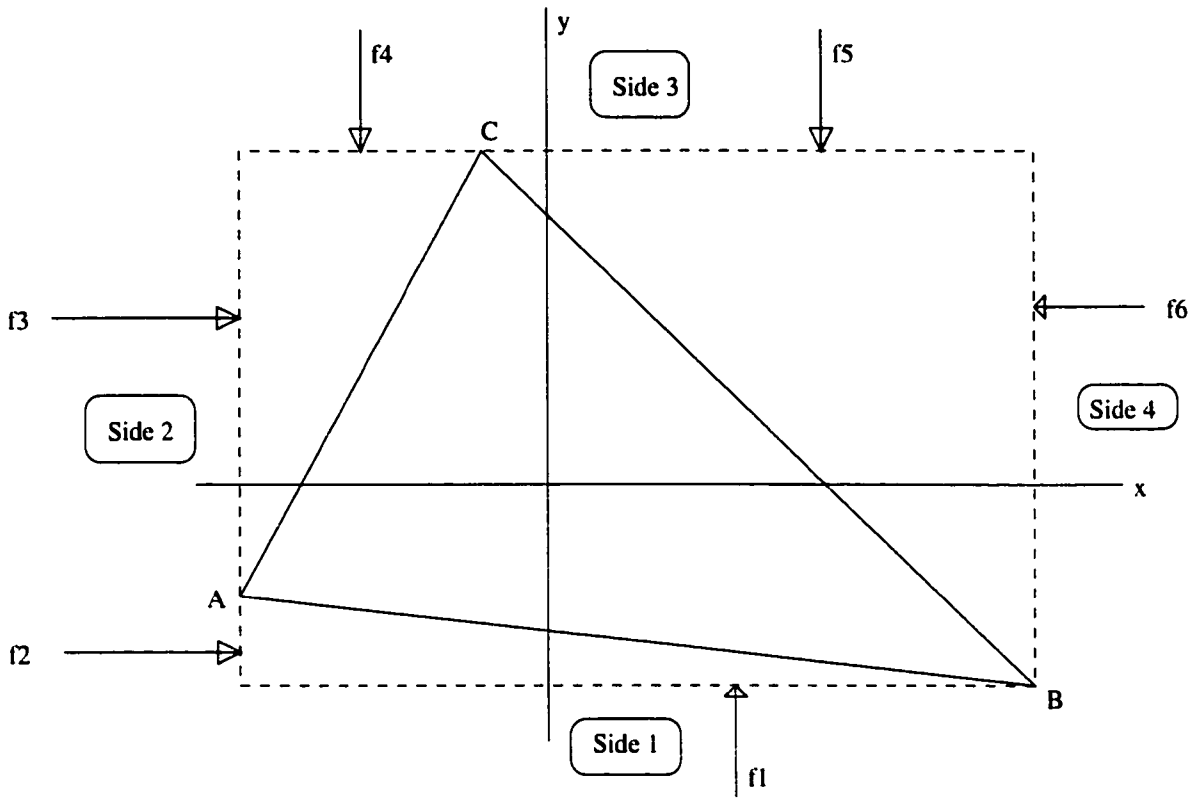

Figure A.1. Triangle, 0° Orientation, Nominal or Ideal Position ($\underline{\beta} = \underline{0}$)

In Figure A.1, the function f1 is given by equation (A.6) and the function f2 is given by equation (A.5).

Similarly, for a given y (or x), one can solve the equations (A.1) and (A.3) for x (or y) and obtain functions f3 and f4 by replacing p1 and q1 in (A.5) and (A.6) with p2 and q2. Again, for a given y (or x), one can solve the equations (A.1) and (A.4) for x (or y) and obtain functions f5 and f6 by replacing p1 and q1 in (A.5) and (A.6) with p3 and q3. Now, one can use the program "der.s" in Appendix B.9 for each of f1 through f6 and obtain the derivative results to enter into the program "der.h" in Appendix B.4.

A.3. TRIANGLE, 45° ORIENTATION

Now, the triangle from Figure A.1 is rotated 45° as shown on Figure A.2. One can use equation (A.1) to find the new positions of the vertices of the triangle in its nominal position. Then the 3 steps illustrated in Section A.2 can be used to get the functions f1 through f6 (and the derivative matrix F) as indicated on Figure A.2.

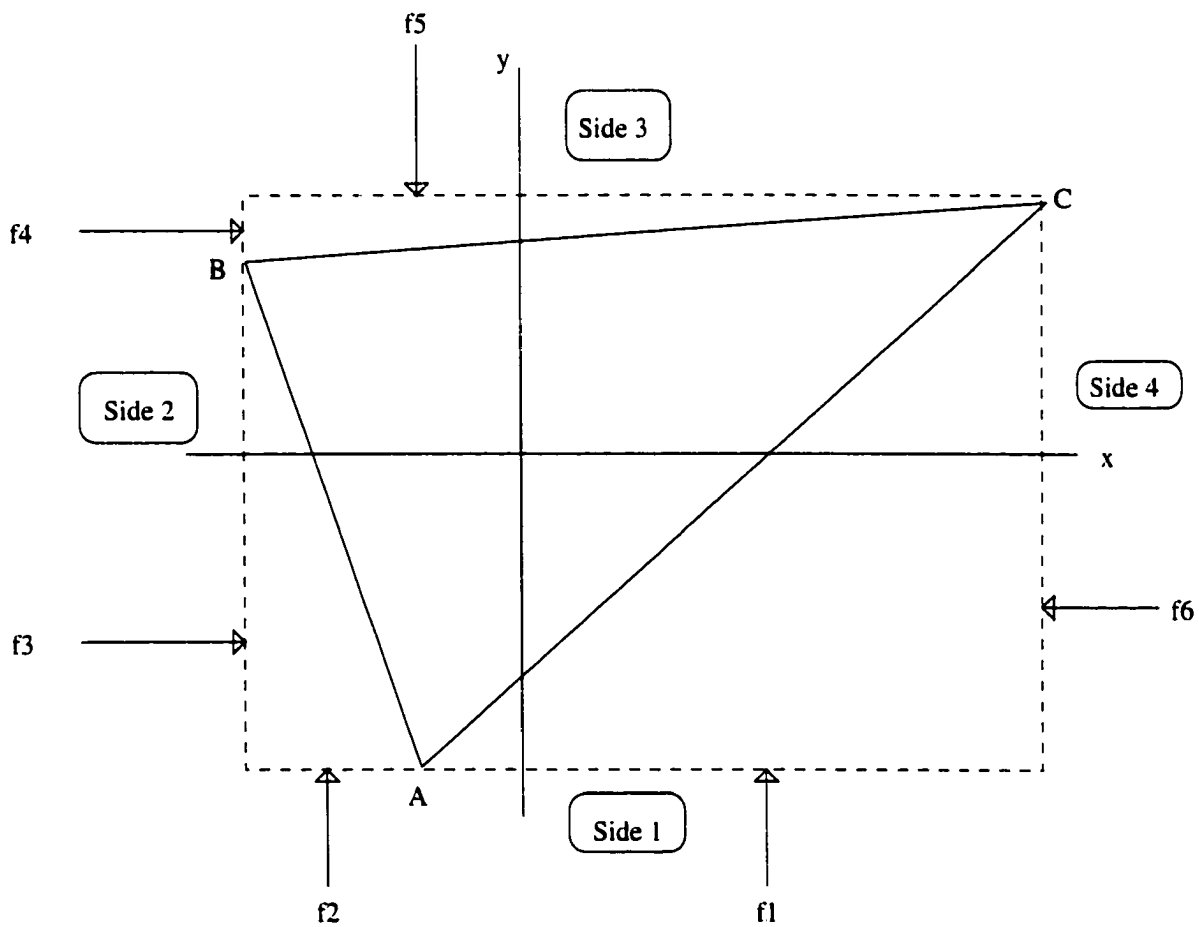

Figure A.2. Triangle, 45° Orientation, Nominal or Ideal Position ($\underline{\beta} = \underline{0}$)

A.4. RECTANGLE, 0° ORIENTATION

The parameters we use for describing this object in its nominal position consist of the length (l_rec) and the width (w_rec) (in a coordinate system with axes parallel to the CMM axes). Then, the center of mass is simply (l_rec/2, w_rec/2) and this is used as the origin of our part coordinate system. With respect to the $\underline{\beta} = \underline{0}$ coordinate system, the 4 shape dependent functions f (see Figure A.3) referred to in Section 3.1 can be obtained using the same procedures as in Sections A.2.1 - A.2.3. These 4 functions are entered into the program der.h (to get the derivative matrix F) as described in Chapter 3.

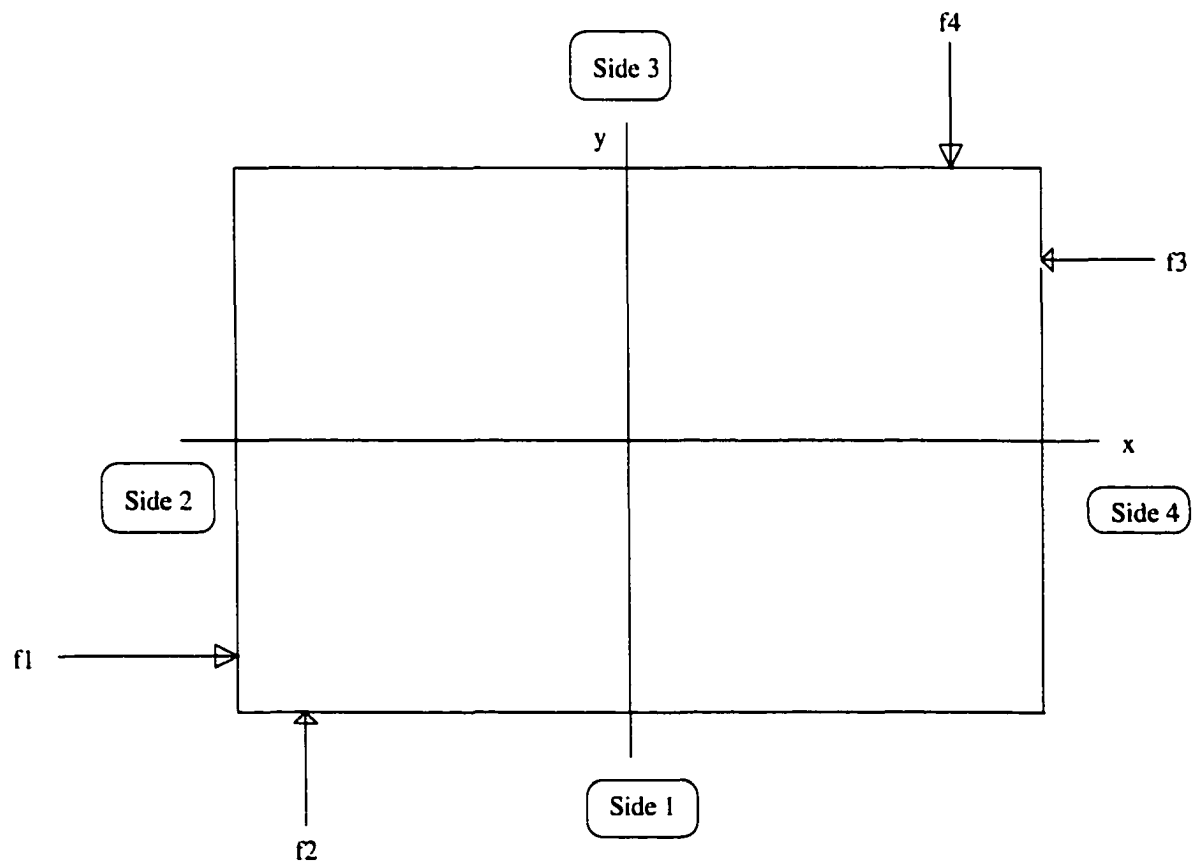

Figure A.3. Rectangle, 0° Orientation, Nominal or Ideal Position ($\underline{\beta} = \underline{0}$)

A.5. RECTANGLE, 45° ORIENTATION

The rectangle of Section A.4 is now rotated 45° and as shown in Figure A.4. Again, the same steps can be used to find the appropriate derivative matrix F to be entered into the program "der.h" for this rectangle, 45° orientation case. In addition, the parameters for the geometry are entered and can be modified in the program "con.h".

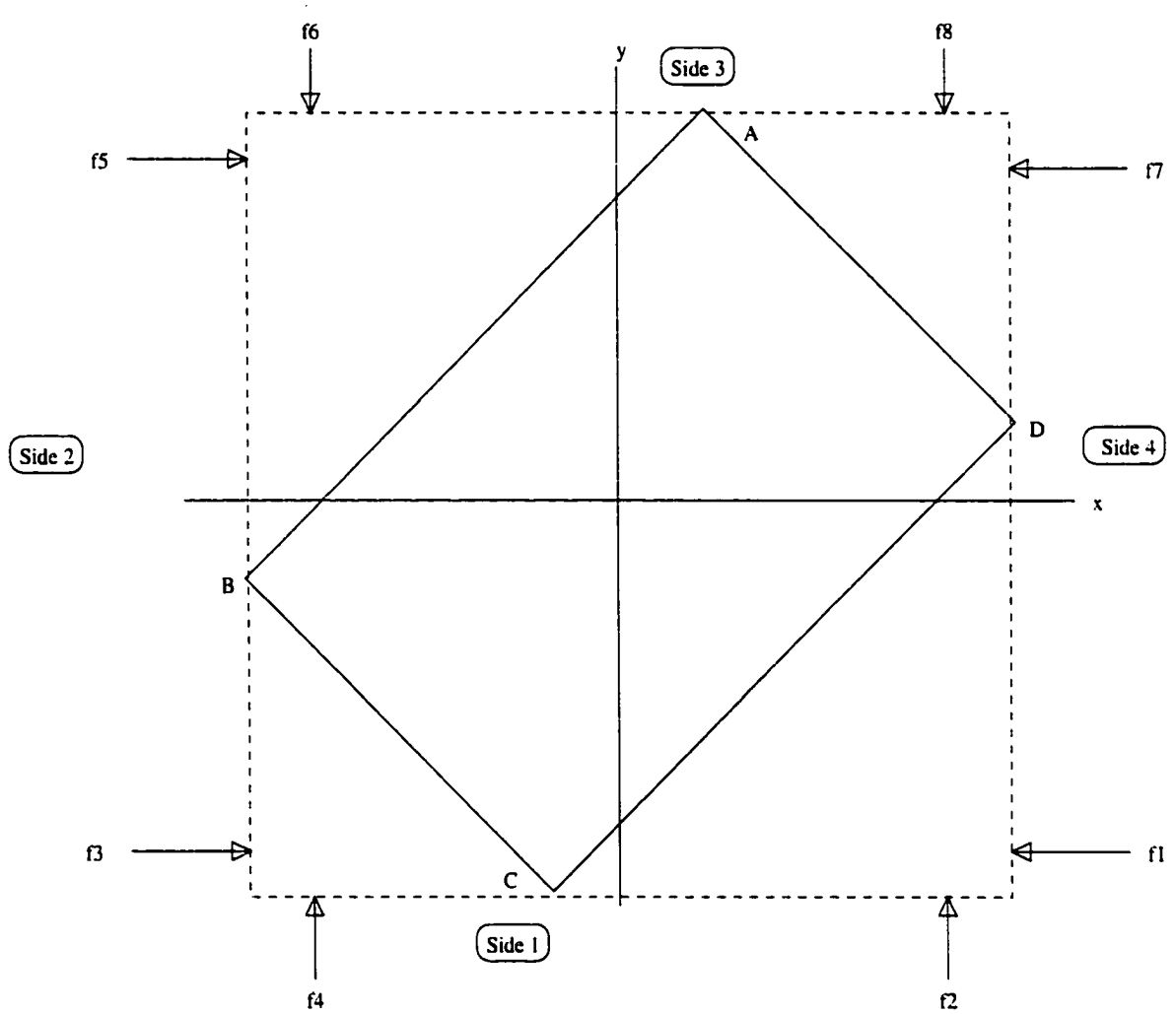

Figure A.4. Rectangle, 45° Orientation, Nominal or Ideal Position ($\underline{\beta} = \underline{0}$)

A.6. HOOD, 0° ORIENTATION

The parameters we use to describe this geometry in nominal position consist of the radii r1, r2 and the angles T1, T2 as shown on Figure A.5 in an arbitrary/convenient coordinate system with axes parallel to the CMM coordinate axes. The center of mass can be found by simple analytical geometry and is then used as the origin of our part coordinate system. Next, the same procedures as before can be applied to this problem.

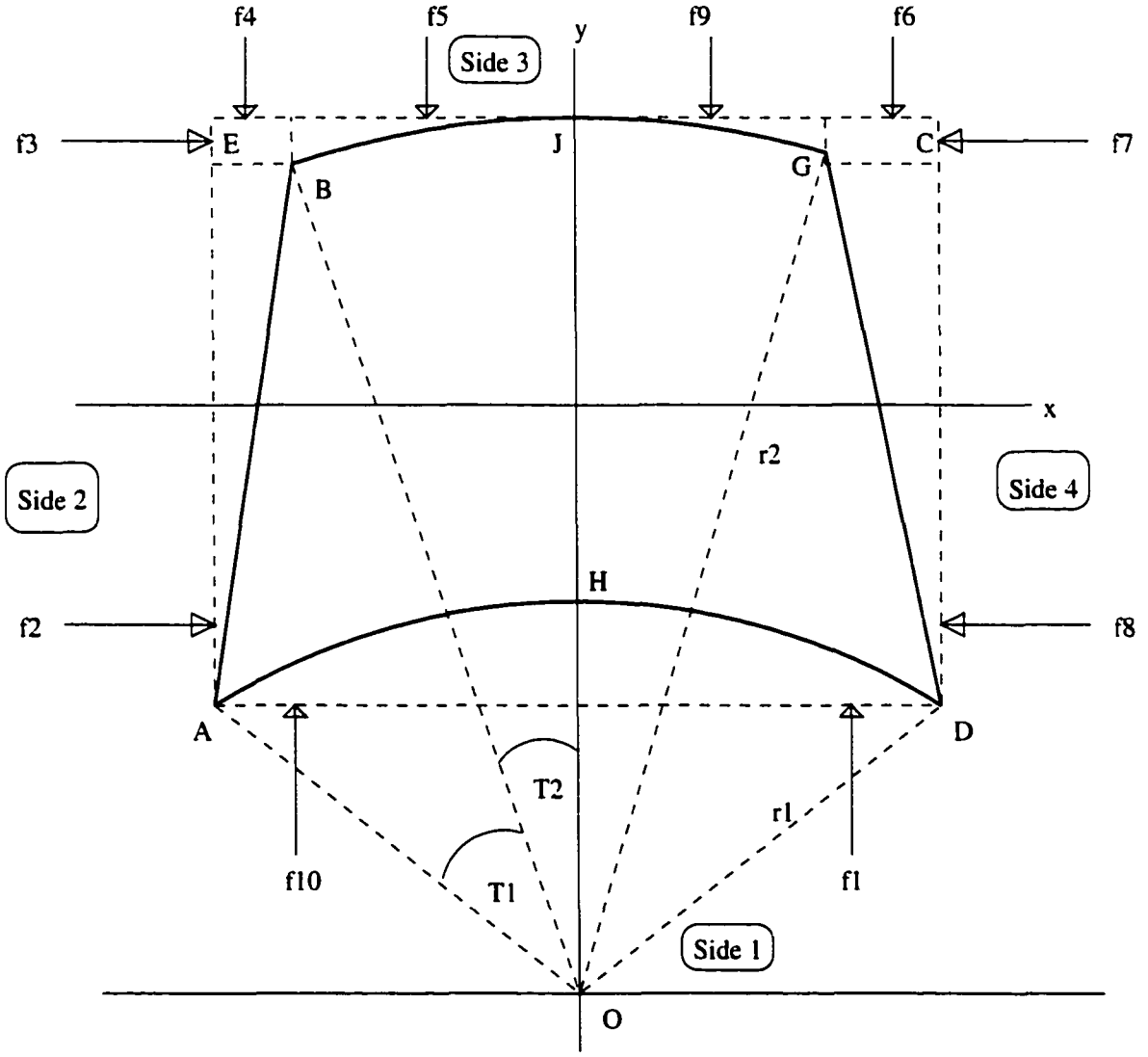

Figure A.5. Hood, 0° Orientation, Nominal or Ideal Position ($\beta = 0$)

A.7. HOOD, 45° ORIENTATION

The hood shape of Section A.6 is rotated 45° and is now shown on Figure A.6 in its nominal position. Again, the same steps produce the appropriate derivative matrix F to be entered to the program "der.h" for this hood shape, 45° orientation case. In addition, the parameters for the geometry are entered and can be modified in the program "con.h".

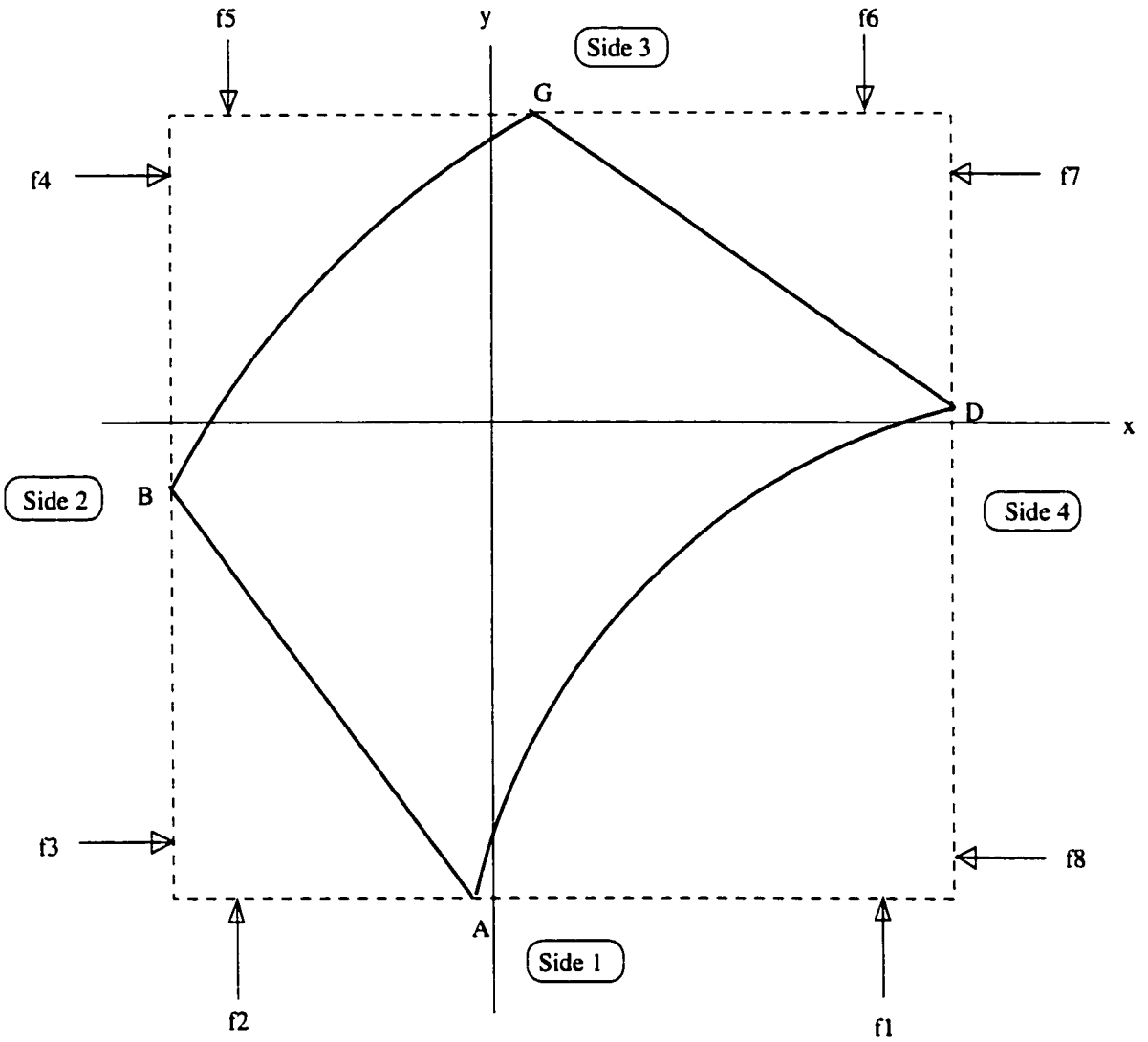

Figure A.6. Hood, 45° Orientation, Nominal or Ideal Position ($\underline{\beta} = \underline{0}$)

A.8. PARABOLAS, 0° ORIENTATION

Here, the object of interest in nominal position is defined by 2 quadratic equations $y = x^2 - z$ and $y = -x^2 + z$. The shape can then by parameterized by the y-intercepts, $\pm z$, as shown on Figure A.7. The center of mass is (0, 0) and is the origin of our $\underline{\beta} = \underline{0}$ part coordinate system. With respect to this coordinate system, the same procedures as before can be applied to this problem.

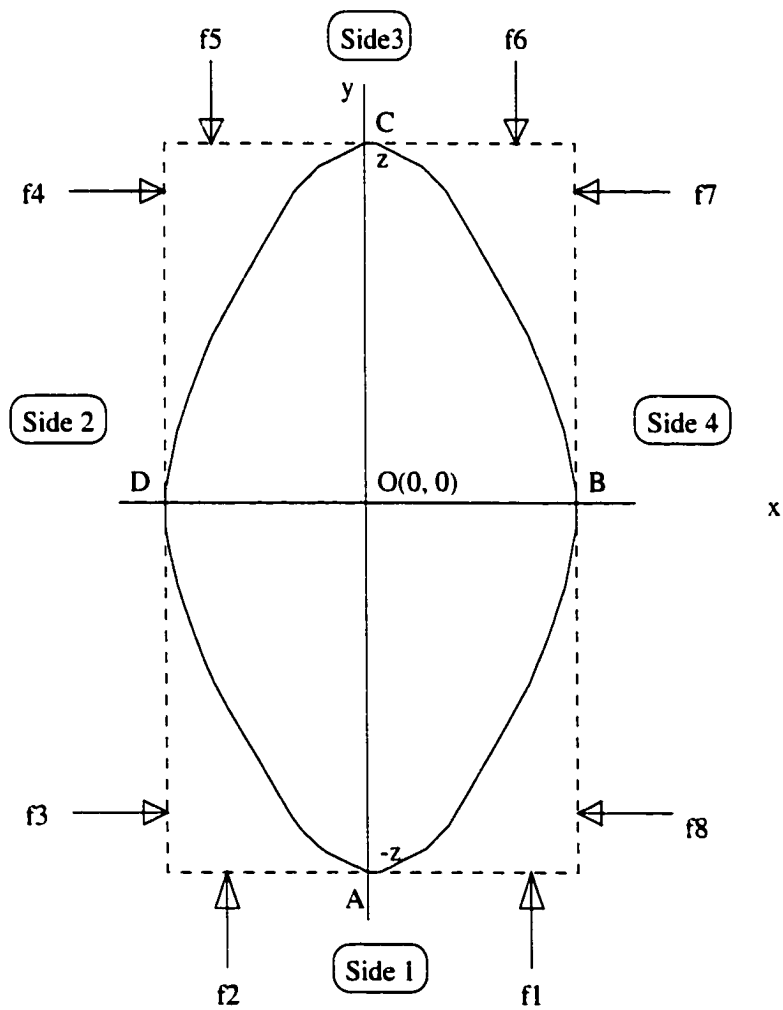

Figure A.7. Parabolas, 0° Orientation, Nominal or Ideal Position ($\underline{\beta} = \underline{0}$)

A.9. PARABOLAS, 45° ORIENTATION

The geometry in Section A.8 is rotated 45° and is now shown on Figure A.7. Again, the same methods can be used to find the appropriate derivative matrix F to be entered in the program "der.h" for the Parabolas, 45° orientation case. In addition, the parameters describing the object are entered and can be modified in the program "con.h".

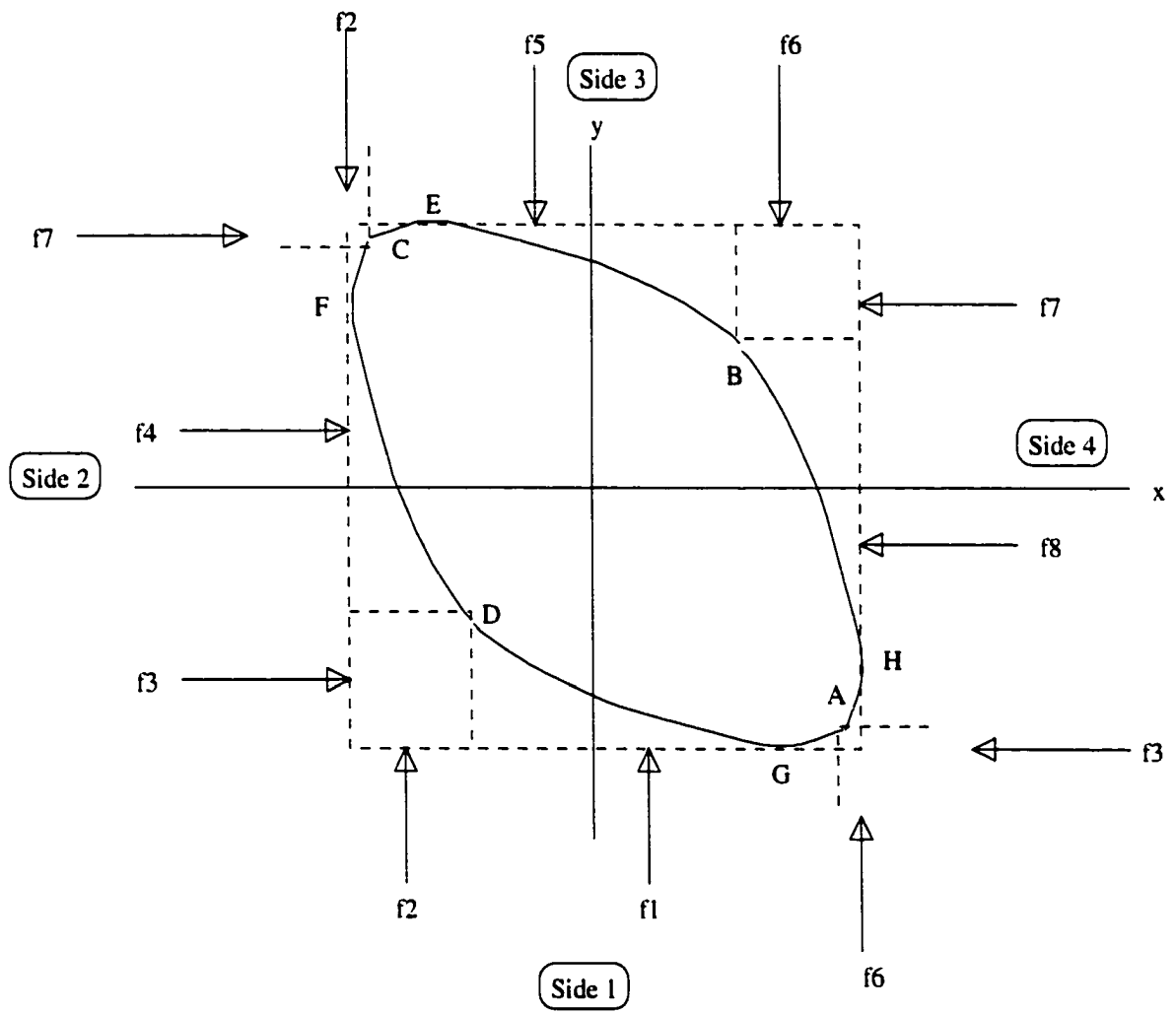

Figure A.8. Parabolas, 45° Orientation, Nominal or Ideal Position ($\underline{\beta} = \underline{0}$)

APPENDIX B. COMPUTER PROGRAMS

Presented here are a number of programs needed to implement the computations described in Chapter 3 for one of our test cases: Triangle, 0° Orientation. For any other shape/orientation, one needs to replace the Triangle, 0° Orientation equations with ones appropriate to the case of interest. We employ several conventions in discussing the programs. The programs themselves (either C programs or S-plus programs) are displayed in the `Courier New font` while all other text is in the standard Times New Roman font. Wherever possible, intuitive names are chosen for variables appearing in the algorithms. Program names ending with ".c" and ".h" are done in c-language and those ending with ".s" are in s-language (Splus).

The programs are partitioned in such a way that they can be easily modified for other 2-dimensional geometries/orientations. The main programs each implement one of the methods of searching for an optimal design (Exhaustive, ONE, or GA) and are named "exhaust.c", "one.c", and "ga.c". Other (partitioned) programs "mat.h", "con.h", and "der.h" are already included/linked to the 3 main programs. Thus, for a different geometry/orientation, one just needs to modify "con.h" and "der.h". Presented here is an example for one particular geometry, namely a triangle. The program "der.s" finds derivatives functions to be entered into the program "der.h". Finally, to plot the graph of $E(Z_n)$ versus n discussed in Section 3.4.5, one can use the program "EZ.s".

The following is a list of the programs and their uses:

cnt.c	computes the total number of designs, for a given m and k,
mat.h	computes the inverse of a real matrix, the weighted trace, and does the detailed work of one-at-a-time computation,
con.h	contains parameters and constants needed for a particular geometry/orientation,
der.h	contains the (shape dependent) derivative functions,
exhaust.c	does an exhaustive search for a best design,
one.c	implements the one-at-a-time search algorithm,
ga.c	implements the genetic algorithm,
EZ.s	plots the graph of $E(Z_n)$ versus n,
der.s	obtains the derivative functions for "der.h" program.

Those programs are presented in Sections B.1 to B.9.

B.1. THE PROGRAM "cnt.c"

As mentioned in Section 3.4, to implement the loops indicated on the left side of (3.5) for counting possible designs, the program "cnt.c" can be used. This program is for m = 80, and k =20. For other m, one can simply change the number 80 in this program to be the number desired. For k points, one can declare more integers at the beginning of the program and modify the number of loops in the existing program to be k loops.

```
/* counting the total number of design possibilities */

#include <stdio.h>
#define m 80
main(){
  int i1, i2, i3, i4, i5, i6, i7, i8, i9, i10;
  int i11, i12, i13, i14, i15, i16, i17, i18, i19, i20;
  long s = 0;

  for(i1=0; i1<m; i1++)
  for(i2=i1; i2<m; i2++)
  for(i3=i2; i3<m; i3++)
  for(i4=i3; i4<m; i4++)
  for(i5=i4; i5<m; i5++)
  for(i6=i5; i6<m; i6++)
  for(i7=i6; i7<m; i7++)
  for(i8=i7; i8<m; i8++)
  for(i9=i8; i9<m; i9++)
  for(i10=i9; i10<m; i10++)
  for(i11=i10; i11<m; i11++)
  for(i12=i11; i12<m; i12++)
  for(i13=i12; i13<m; i13++)
  for(i14=i13; i14<m; i14++)
  for(i15=i14; i15<m; i15++)
  for(i16=i15; i16<m; i16++)
  for(i17=i16; i17<m; i17++)
  for(i18=i17; i18<m; i18++)
  for(i19=i18; i19<m; i19++)
  for(i20=i19; i20<m; i20++)
              s++;
  printf("total number of designs to check: %ld\n", s);
}
```

B.2. THE PROGRAM "mat.h"

The following program does not need to be modified for other geometries/orientations. Basically, it computes the inverse of a real matrix, the weighted trace, and does the detailed work of the one-at-a-time search algorithm.

```
/* equally spaced length-c-sequence from a to b(if a<b) or b to a(if
a>b)*/
void seq (double a, double b, int c, double *out){
  int i;
  double ss;
  ss = (b-a)/(c-1.0);
  if (a < b)
    for (i = 0; i < c; i++) *(out+i) = a + i * ss;
  else{
```

```
    ss = -ss;
    for (i = 0; i < c; i++) *(out+i) = a - i * ss;}
}

/* Rank a real vector
   rv is the real vector to be ranked.
   M2 is the size of rv.
   order is either ascending or decending.
   irank is the rank for output*/
void rank(double *rv, int m2, char order, int *irank){
  int m1=1, ifail=0;
  double m01daf_(), m01zaf_();
  m01daf_(rv, &m1, &m2, &order, irank, &ifail);
  m01zaf_(irank, &m1, &m2, &ifail);
  for (m1 = 0; m1 < m2; m1++) irank[m1]--;
}

/* Invert A of size_A.
   size is defined by macro and must be >= size_A
   define size as the maximum of those matrices to
   be inverted

   if A is invertible, then return *info=0, and A<-inverse(A).
           otherwise,       return *info=1. and A<-LU(ignored)
         this is tested!!!

#define size 3
void inv (double *A, int size_A, int *info_p){
  int mm, n1, lda, lwork = size * size;
  int  ipiv[size];
  double Work[size*size];
  double f07adf_(), f07ajf_();
  mm=size_A; n1=size_A; lda=size_A;
  f07adf_(&mm, &n1, A, &lda, ipiv, info_p);
  if (*info_p == 0) f07ajf_(&n1, A, &lda, ipiv, Work, &lwork, info_p);
  else *info_p=1;
}
*/


/* invert a real symmetric matrix stored as a vector of
   the upper triangular part -- A. size_A is the order of
   that matrix.
   if A is invertible, then return *info_p=0, A<-vech(inv A)
   otherwise return *info_p=1
*/
#define size 3
void inv_sym (double *A, int size_A, int *info_p){
  char uplo='L';/* note the storage diff between C and Fortran */
  int ipiv[size];
  double work[size];
  double f07pdf_(), f07pjf_();

  f07pdf_(&uplo, &size_A, A, ipiv, info_p);
  if (*info_p ==0) f07pjf_(&uplo, &size_A, A, ipiv, work, info_p);
```

```
  else *info_p = 1;
}

/* compute weighted trace.
   F is df/d(u,v,t), W is the weight
   if F'*F invertible return weighted trace.
   otherwise return 0.*/
double wtrace(double F[k][3], double W[3]){
  int r, c, i, cnt, info, *info_p;
  double C[6];

  for (r = 0; r < 6; r++) C[r]=0.;
  info_p = &info;
  cnt=0;
  for ( r = 0; r < 3; r++)
    for (c = r; c < 3; c++){
      for (i = 0; i < k; i++) C[cnt] += F[i][r]*F[i][c];
      cnt++;}
  inv_sym (C, 3, info_p);
  if (*info_p == 0) return(W[0]*C[0]+W[1]*C[3]+W[2]*C[5]);
  else return (0.);
}

/* one at a time search */
double one_a_time(int *i, int *ind, double *W, double *dat){
  int j1, j2, dex[k];
  double t1, t2;
  double x[k];
  double awt_rec();
  for (j1 = 0; j1 < k-1; j1++) {
    x[j1]=dat[*(i+j1)]; dex[j1]=ind[*(i+j1)];}

  for (j1 = k-1; j1 >= 0; j1--){
    *(i+j1) = 0;
    x[j1] = dat[0]; dex[j1]=ind[0];
    t1 = awt_rec(x, dex, W);
    for (j2 = 1; j2 < m; j2++){
      x[j1] = dat[j2]; dex[j1]=ind[j2];
      t2 = awt_rec(x, dex, W);
      if (t1>t2) {*(i+j1)=j2; t1=t2;}
    }
    x[j1]=dat[*(i+j1)]; dex[j1]=ind[*(i+j1)];
  }
  return(t1);
}

/* two at a time search */
double two_a_time(int *i, int *ind, double *W, double *dat){
  int j1, j2, j3, j4, dex[k];
  double t1, t2;
  double x[k];
  double awt_rec();
  for (j1 = 0; j1 < k-2; j1++) {
    x[j1]=dat[*(i+j1)]; dex[j1]=ind[*(i+j1)];}
```

```
  for (j1 = k-1; j1 >= 0; j1--){
    *(i+j1) = 0;
    x[j1] = dat[0]; dex[j1]=ind[0];
    for (j2 = k-2; j2 >= 0; j2--){
      *(i+j2) = 0;
      x[j2] = dat[0]; dex[j2]=ind[0];
      t1 = awt_rec(x, dex, W);
      for (j3 = 1; j3 < m; j3++){
      x[j1] = dat[j3]; dex[j1]=ind[j3];
      for (j4 = 1; j4 < m; j4++){
        x[j2] = dat[j4]; dex[j2]=ind[j4];
        t2 = awt_rec(x, dex, W);
        if (t1>t2) {
          *(i+j1)=j3;
          *(i+j2)=j4;
          t1=t2;}
      }
      }
      x[j1]=dat[*(i+j1)]; dex[j1]=ind[*(i+j1)];
      x[j2]=dat[*(i+j2)]; dex[j2]=ind[*(i+j2)];
    }
  }
  return(t1);
}
```

B.3. THE PROGRAM "con.h"

This is the program where one enters those constants needed to describe a particular geometry/orientation. For the purpose of example, this is the particular program "con.h" for a Triangle Shape, and 0° Orientation. The parameters chosen for this triangle shape are the 3 points (x, y) specified with respect to an arbitrary/convenient origin. One can change m , k, n_1 (number of random starts desired for ONE), seed (random seed choice), etc. Other comments are in the program itself.

```
#define k 16 /* number of affordable points or msmts */
#define m 40 /* number of possible slots */
#define X1 (-15.)/* x-coordinate for 1st point */
#define Y1 (- 5.) /* y-coordinate for 1st point */
#define X2   30. /* x-coordinate for 2nd point */
#define Y2 (-10.)/* y-coordinate for 2nd point */
#define X3 (- 1.) /* x-coordinate for 3rd point */
#define Y3   20. /* y-coordinate for 3rd point */
```

```
#define Xbar ((X1+X2+X3)/3.) /* x-coordinate for the Center of Mass */
#define Ybar ((Y1+Y2+Y3)/3.) /* y-coordinate for the Center of Mass */
#define XA (X1-Xbar) /* x-coordinate for 1st point wrt the CM */
#define XB (X2-Xbar) /* x-coordinate for 2nd point wrt the CM */
#define XC (X3-Xbar) /* x-coordinate for 3rd point wrt the CM */
#define YA (Y1-Ybar) /* y-coordinate for 1st point wrt the CM */
#define YB (Y2-Ybar) /* y-coordinate for 2nd point wrt the CM */
#define YC (Y3-Ybar) /* y-coordinate for 3rd point wrt the CM */
#define p1  (Y2-Y1)/(X2-X1)
#define p2  (Y3-Y1)/(X3-X1)
#define p3  (Y3-Y2)/(X3-X2)
#define q1  ((Xbar-X1)*(Y2-Y1)/(X2-X1)+Y1-Ybar)
#define q2  ((Xbar-X1)*(Y3-Y1)/(X3-X1)+Y1-Ybar)
#define q3  ((Xbar-X2)*(Y3-Y2)/(X3-X2)+Y2-Ybar)
#define WT_SIZE /*8855*/ 2220075 /* using test/cnt.s */
#define w1 1. /* the weight omega1 */
#define w2 1. /* the weight omega2 */
#define MAX(x,y) (x)>(y) ? (x):(y)
#define jnk MAX(pow(XA,2.)+pow(YA,2.),pow(XB,2.)+pow(YB,2.))
#define w3 MAX(jnk,pow(XC,2.)+pow(YC,2.)) /*(furthest distance from
CM)^2*/
#define BIG 1.e6
#define n_1 1500 /* number of starts for one-at-a-time */
#define n_2 50 /* number of starts for two-at-a-time */
#define seed 100 /* random number seed */
```

B.4. THE PROGRAM "der.h"

For the purpose of example, this is the "der.h" program for Triangle Shape, 0° orientation. There are 2 options (horizontally/vertically) to "touch" each of the 3 sides. Hence there are 6 "switches" for the derivative functions needed for this program. Each "switch" refers to a different "derivative function" named df1, ... ,df6 in this program. For a different shape, one can just adjust the "switch" number (indicated by "`case i:`") in this program to be as many derivative functions (indicated by `/* dfi/d(u,v,t) */` in this program) as needed.

```
/* make data and corresponding indices telling which derivative to use */

void get_dat(double *dat, int *ind){
  int i0;
```

```
/* set up the potential points */
  seq (XA+.5, XB-.5, (m/4), dat); /* side 1 */
  seq (XA+.5, XB-.5, (m/4), dat+m/4*2); /* side 3 */
  seq (YB+.5, YC-.5, (m/4), dat+m/4); /* side 2 */
  seq (YB+.5, YC-.5, (m/4), dat+m/4*3); /* side 4 */

/* set up the indices */
  for (i0 = 0; i0 < m/4; i0++)
      ind[i0] = 1;
  for (i0 = m/4; i0 < m/4*2; i0++){
    if(dat[i0]<=YA)
      ind[i0] = 2;
    else ind[i0] = 3;}
  for (i0 = m/4*2; i0 < m/4*3; i0++){
    if(dat[i0]<=XC)
      ind[i0] = 4;
    else ind[i0] = 5;}
  for (i0 = m/4*3; i0 < m; i0++)
      ind[i0] = 6;

}

/* triangular shape
   compute average weighted trace for 3*3*3 u, v, t
   k is the number of affordable points,
   X is the points. ind indicates at which side.
   W is the weights.
   if one of the weighted trace is 0., return a big value*/
  double awt_rec(double x[k], int ind[k], double W[3]){
  int iu, iv, it, ik;
  double F[k][3];
  double u[3]={-.25, 0., .25}, v[3]={-.25, 0., .25}, t[3]={-.1, 0., .1};
  double result, ave = 0.;
  void df1(), df2(), df3(), df4(), df5(), df6(), df7(), df8();
  for (iu = 0; iu < 3; iu++){
    for (iv = 0; iv < 3; iv++){
      for (it = 0; it < 3; it++){
      for (ik = 0; ik < k; ik++){
        switch(ind[ik]){
        case 1: df1(x[ik], u[iu], v[iv], t[it], F[ik]); break;
        case 2: df2(x[ik], u[iu], v[iv], t[it], F[ik]); break;
        case 3: df3(x[ik], u[iu], v[iv], t[it], F[ik]); break;
        case 4: df4(x[ik], u[iu], v[iv], t[it], F[ik]); break;
        case 5: df5(x[ik], u[iu], v[iv], t[it], F[ik]); break;
        case 6: df6(x[ik], u[iu], v[iv], t[it], F[ik]); break;
        default: printf("something wrong with awt_rec\n"); break;}
      }
      result = wtrace (F, W);
      if (result <= 0.) {ave = BIG; goto end;}
      else ave += result/27.;
      }
    }
  }
 end: return (ave);
}
```

```
/* df1/d(u,v,t) */
void df1(double X, double u, double v, double t, double *F){
  double expr2, expr3, expr4, expr5, expr7;
  double expr8, expr10, expr13;
  expr2 = sin(t);
  expr3 = q1 * expr2;
  expr4 = (X - u) + expr3;
  expr5 = cos(t);
  expr7 = expr2 + (p1 * expr5);
  expr8 = expr4 * expr7;
  expr10 = expr5 - (p1 * expr2);
  expr13 = q1 * expr5;
  *F = - (expr7/expr10);
  *(F+1) = 1. ;
  *(F+2) = ((((expr13 * expr7) + (expr4 * expr10))/
                  expr10) + ((expr8 * expr7)/pow(expr10,2.))) - expr3;
}

/* df2/d(u,v,t) */
void df2(double X, double u, double v, double t, double *F){
  double expr2, expr3, expr4, expr5, expr7;
  double expr8, expr10, expr13;
  expr2 = cos(t);
  expr3 = q1 * expr2;
  expr4 = (X - v) - expr3;
  expr5 = sin(t);
  expr7 = expr2 - (p1 * expr5);
  expr8 = expr4 * expr7;
  expr10 = expr5 + (p1 * expr2);
  expr13 = q1 * expr5;
  *F = 1.;
  *(F+1) = - (expr7/expr10);
  *(F+2) = ((((expr13 * expr7) - (expr4 * expr10))/
                  expr10) - ((expr8 * expr7)/pow(expr10,2.))) - expr3;
}

/* df3/d(u,v,t) */
void df3(double X, double u, double v, double t, double *F){
  double expr2, expr3, expr4, expr5, expr7;
  double expr8, expr10, expr13;
  expr2 = cos(t);
  expr3 = q2 * expr2;
  expr4 = (X - v) - expr3;
  expr5 = sin(t);
  expr7 = expr2 - (p2 * expr5);
  expr8 = expr4 * expr7;
  expr10 = expr5 + (p2 * expr2);
  expr13 = q2 * expr5;
  *F = 1.;
  *(F+1) = - (expr7/expr10);
  *(F+2) = ((((expr13 * expr7) - (expr4 * expr10))/
                  expr10) - ((expr8 * expr7)/pow(expr10,2.))) - expr3;
}
```

```
/* df4/d(u,v,t) */
void df4(double X, double u, double v, double t, double *F){
  double expr2, expr3, expr4, expr5, expr7;
  double expr8, expr10, expr13;
  expr2 = sin(t);
  expr3 = q2 * expr2;
  expr4 = (X - u) + expr3;
  expr5 = cos(t);
  expr7 = expr2 + (p2 * expr5);
  expr8 = expr4 * expr7;
  expr10 = expr5 - (p2 * expr2);
  expr13 = q2 * expr5;
  *F = - (expr7/expr10);
  *(F+1) = 1. ;
  *(F+2) = ((((expr13 * expr7) + (expr4 * expr10))/
                  expr10) + ((expr8 * expr7)/pow(expr10,2.))) - expr3;
}

/* df5/d(u,v,t) */
void df5(double X, double u, double v, double t, double *F){
  double expr2, expr3, expr4, expr5, expr7;
  double expr8, expr10, expr13;
  expr2 = sin(t);
  expr3 = q3 * expr2;
  expr4 = (X - u) + expr3;
  expr5 = cos(t);
  expr7 = expr2 + (p3 * expr5);
  expr8 = expr4 * expr7;
  expr10 = expr5 - (p3 * expr2);
  expr13 = q3 * expr5;
  *F = - (expr7/expr10);
  *(F+1) = 1. ;
  *(F+2) = ((((expr13 * expr7) + (expr4 * expr10))/
                  expr10) + ((expr8 * expr7)/pow(expr10,2.))) - expr3;
}

/* df6/d(u,v,t) */
void df6(double X, double u, double v, double t, double *F){
  double expr2, expr3, expr4, expr5, expr7;
  double expr8, expr10, expr13;
  expr2 = cos(t);
  expr3 = q3 * expr2;
  expr4 = (X - v) - expr3;
  expr5 = sin(t);
  expr7 = expr2 - (p3 * expr5);
  expr8 = expr4 * expr7;
  expr10 = expr5 + (p3 * expr2);
  expr13 = q3 * expr5;
  *F = 1.;
  *(F+1) = - (expr7/expr10);
  *(F+2) = ((((expr13 * expr7) - (expr4 * expr10))/
                  expr10) - ((expr8 * expr7)/pow(expr10,2.))) - expr3;
}
```

B.5. THE PROGRAM "exhaust.c"

This exhaustive search program is for k = 4 points. For k other than 4, this program can be modified by declaring more integers (i0, i1, ..., ik) at the beginning of the program and adding more loops and making the necessary modifications in the marked part "/* Get the Data and Indices */". The number of candidate paths, m, is entered in the program "con.h".

```
/* exhaust search */

#include <stdio.h>
#include <math.h>
#include "con.h"    /* contains project constants */
#include "../mat.h"    /* supporting routines */
#include "der.h"

main(){
  int i0, i1, i2, i3, i4;
  int cnt=0;
  int ind[m];     /* index of potential points
                        3            11--15
                      2 4          6--10   16--20
                        1              1---5
                */
  int i[WT_SIZE][k];  /* index of affordable points */
  int dex[k];         /* which sides of the affordable points */
  int WT_RANK[WT_SIZE]; /* rank of the trace */
  double dat[m]; /* potential points */
  double x[k]; /* affordable points */
  double W[3]={w1, w2};/* weights */
  double WT[WT_SIZE]; /* all possible trace */
  W[2]=w3;

/* Get the Data and Indices */
  get_dat (dat, ind);

  for (i1 = 0; i1 < m; i1++){
    for (i2 = i1; i2 < m; i2++){
      for (i3 = i2; i3 < m; i3++){
      for (i4 = i3; i4 < m; i4++){
        x[0]=dat[i1]; x[1]=dat[i2]; x[2]=dat[i3]; x[3]=dat[i4];
        i[cnt][0]=i1; i[cnt][1]=i2;
        i[cnt][2]=i3; i[cnt][3]=i4;
        dex[0]=ind[i1]; dex[1]=ind[i2]; dex[2]=ind[i3];dex[3]=ind[i4];
        WT[cnt] = awt_rec(x, dex, W);
        cnt++;}
```

```
      }
    }
  }
  rank (WT, WT_SIZE, 'a', WT_RANK);
  printf("Exausted Search\n");
  printf("m=%d k=%d\n", m, k);
  for (i0 = 0; i0 < 10; i0++){
    printf("Points:\t");
    for (i1 = 0; i1 < k; i1++) printf("%d\t", i[WT_RANK[i0]][i1]+1);
    printf("\nSides\t");
    for (i1 = 0; i1 < k; i1++) printf("%d\t", ind[i[WT_RANK[i0]][i1]]);
    printf("%f\n", WT[WT_RANK[i0]]);}
  printf("\n");
}
```

B.6. THE PROGRAM "one.c"

The following program does not need to be modified for other geometries/orientations.

```
/* one at a time search */

#include <stdio.h>
#include <math.h>
#include "con.h"    /* contains project constants */
#include "../mat.h"    /* supporting routines */
#include "der.h"
main(){
  int i0, i1, tmp;
  int mm, nn;    /* end points for uniform distribution */
  int ind[m];     /* index of potential points
                     3              11--15
                   2 4         6--10    16--20
                     1              1---5
              */
  int i[n_1][k];  /* index of affordable points */
  int WT_RANK[n_1]; /* rank of the trace */

  double dat[m]; /* potential points */
  double x[k]; /* affordable points */
  double W[3]={w1, w2};/* weights */
  double WT[n_1]; /* the smallest trace for n_1 random starts*/
  double g05cbf_(), g05dyf_();
  W[2] = w3;

/* Get the Data and Indices */
  get_dat (dat, ind);
/* set the seed */
  i0 = seed;
```

```
  g05cbf_(&i0);

/* i0 loops over n_1 random starts */
  mm = 0; nn = m - 1;
  for (i0 = 0; i0 < n_1; i0++){
    for (i1 = 0; i1 < k-1; i1++)
      i[i0][i1] = g05dyf_(&mm, &nn);/* generate uniform(mm, nn) */
    WT[i0] = one_a_time(i[i0], ind, W, dat);}
  rank (WT, n_1, 'a', WT_RANK);
  printf("One at a Time Search With %d Random Starts\n", n_1);
  printf("m=%d k=%d\n", m, k);
  for (i0 = 0; i0 < n_1; i0++) {
    printf("Points:\t");
    for (i1 = 0; i1 < k; i1++) printf("%d\t", i[WT_RANK[i0]][i1]+1);
    printf("\nSides\t");
    for (i1 = 0; i1 < k; i1++) printf("%d\t", ind[i[WT_RANK[i0]][i1]]);
    printf("%f\n", WT[WT_RANK[i0]]);}
  printf("\n");

}
```

B.7. THE PROGRAM "ga.c"

The following program does not need to be modified for other geometries/orientations. The parameters (such as POPSIZE, MAXGENS, NVARS, PXOVER, PMUTATION, n_g) can be changed here (as shown in the beginning of this program). The choice of constants such as m, k, seed, etc. can be made in the program "con.h".

```
/*****************************************************************/
/* This DISCRETE genetic algorithm is to be used with the        */
/* programs "math.h", "der.h" and "con.h"                        */
/* and also the NAG library as well as the MATH library          */
/*****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "con.h"
#include "../mat.h"
#include "der.h"

/* GA parameters of interest */

#define POPSIZE 8                 /* population size = #of designs*/
#define MAXGENS 40                /* max. number of generations */
```

```
#define NVARS k                    /* # of k measurements */
#define PXOVER 0.8                 /* probability of crossover */
#define PMUTATION 0.15             /* probability of mutation */
#define TRUE 1
#define FALSE 0
#define n_g 2500                /* no. of looping random starts */

int generation;                    /* current generation no. */
int cur_best;                      /* best individual */

struct genotype /* genotype (GT), a member of the population */
{
  int gene[NVARS];              /* a string of variables */
  double fitness;               /* GT's fitness (1/this = Wt.Trace)*/
  int upper[NVARS];             /* GT's variables upper bound */
  int lower[NVARS];             /* GT's variables lower bound */
  double rfitness;              /* relative fitness */
  double cfitness;              /* cumulative fitness */
};

struct genotype population[POPSIZE+1];    /* population */
struct genotype newpopulation[POPSIZE+1]; /* new population; */
                                          /* replaces the */
                                          /* old generation */

/* declare 2 variables to be used with der.h */
int Ind[m];
double Dat[m];

/* Declaration of functions */

void initialize(void);
int randval(int, int);
void evaluate(void);
void keep_the_best(void);
void elitist(void);
void Select(void);
void crossover(void);
void Xover(int,int);
void swap(int *, int *);
void mutate(void);



/*************************************************************/
/* Initialization function: Initializes the values of genes    */
/* within the measurement's bounds (i.e. within  1 & m)        */
/*************************************************************/

void initialize(void)
{
int i, j;
int lbound=0, ubound=m-1;

/* initialize variables/measurements within the bounds */
```

```
for (i = 0; i < NVARS; i++)
      {
      for (j = 0; j < POPSIZE; j++)
            {
            population[j].fitness = 0;
            population[j].rfitness = 0;
            population[j].cfitness = 0;
            population[j].lower[i] = lbound;
            population[j].upper[i]= ubound;
            population[j].gene[i] = randval(population[j].lower[i],
                                    population[j].upper[i]);
            }
      }

}

/*************************************************************/
/* Random value generator: Generates a value within bounds */
/*************************************************************/

int randval(int low, int high)
{
int val;
int lb = 0, ub = m-1;
double g05dyf_();
val = g05dyf_(&lb, &ub);
return(val);
}

/*************************************************************/
/* Evaluation function: is the fitness function              */
/* The smaller the AWT, the better fitness it will be        */
/*************************************************************/

void evaluate(void)
{
int mem;
int i;
int der[k]; /* used to be dex[k] in rec.all.c */
double W[3] = {w1, w2};
double x[NVARS];
W[2] = w3;

for (mem = 0; mem < POPSIZE; mem++)
      {
      for (i = 0; i < NVARS; i++){
            x[i] = Dat[population[mem].gene[i]];
          der[i] = Ind[population[mem].gene[i]];
        }
      population[mem].fitness = 1/awt_rec(x, der, W);
    }
}

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

```
/* Keep_the_best function: keep track of the best member of    */
/* the population                                              */
/*************************************************************/

void keep_the_best()
{
int mem;
int i;
cur_best = 0; /* stores the index of the best individual */

for (mem = 0; mem < POPSIZE; mem++)
      {
      if (population[mem].fitness > population[POPSIZE].fitness)
            {
            cur_best = mem;
            population[POPSIZE].fitness = population[mem].fitness;
            }
      }
/* once the best member in the population is found, copy the genes */
for (i = 0; i < NVARS; i++)
      population[POPSIZE].gene[i] = population[cur_best].gene[i];
}



/*************************************************************/
/* Elitist function: To make sure the generation gets better &  */
/* better                                                      */
/*************************************************************/

void elitist()
{
int i;
double best, worst;             /* best and worst fitness values */
int best_mem, worst_mem; /* indexes of the best and worst member */

best = population[0].fitness;
worst = population[0].fitness;
for (i = 0; i < POPSIZE - 1; ++i)
      {
      if(population[i].fitness > population[i+1].fitness)
            {
            if (population[i].fitness >= best)
                  {
                  best = population[i].fitness;
                  best_mem = i;
                  }
            if (population[i+1].fitness <= worst)
                  {
                  worst = population[i+1].fitness;
                  worst_mem = i + 1;
                  }
            }
      else
            {
```

```
            if (population[i].fitness <= worst)
                  {
                  worst = population[i].fitness;
                  worst_mem = i;
                  }
            if (population[i+1].fitness >= best)
                  {
                  best = population[i+1].fitness;
                  best_mem = i + 1;
                  }
            }
      }
/* if best individual from the new population is better than */
/* the best individual from the previous population, then    */
/* copy the best from the new population; else replace the   */
/* worst individual from the current population with the     */
/* best one from the previous generation                     */

if (best >= population[POPSIZE].fitness)
    {
    for (i = 0; i < NVARS; i++)
       population[POPSIZE].gene[i] = population[best_mem].gene[i];
    population[POPSIZE].fitness = population[best_mem].fitness;
    }
else
    {
    for (i = 0; i < NVARS; i++)
       population[worst_mem].gene[i] = population[POPSIZE].gene[i];
    population[worst_mem].fitness = population[POPSIZE].fitness;
    }
}
/*************************************************************/
/* Selection function: select the member according to the    */
/* fitness of each member                                    */
/*************************************************************/

void Select(void)
{
int mem, i, j;
double sum = 0;
double p;

/* find total fitness of the population */
for (mem = 0; mem < POPSIZE; mem++)
      {
      sum += population[mem].fitness;
      }

/* calculate relative fitness */
for (mem = 0; mem < POPSIZE; mem++)
      {
      population[mem].rfitness =  population[mem].fitness/sum;
      }
population[0].cfitness = population[0].rfitness;
```

```
/* calculate cumulative fitness */
for (mem = 1; mem < POPSIZE; mem++)
      {
      population[mem].cfitness =  population[mem-1].cfitness +
                        population[mem].rfitness;
      }

/* finally select survivors using cumulative fitness. */

for (i = 0; i < POPSIZE; i++)
      {
      p = rand()%1000/1000.0;
      if (p < population[0].cfitness)
            newpopulation[i] = population[0];
      else
            {
            for (j = 0; j < POPSIZE;j++)
                  if (p >= population[j].cfitness &&
                              p<population[j+1].cfitness)
                        newpopulation[i] = population[j+1];
            }
      }
/* once a new population is created, copy it back */

for (i = 0; i < POPSIZE; i++)
      population[i] = newpopulation[i];
}

/*************************************************************/
/* Crossover selection: selects two parents that take part in  */
/* the crossover. Implements a single point crossover          */
/*************************************************************/

void crossover(void)
{
int i, mem, one;
int first  =  0; /* count of the number of members chosen */
double x;

for (mem = 0; mem < POPSIZE; ++mem)
      {
      x = rand()%1000/1000.0;
      if (x < PXOVER)
            {
            ++first;
            if (first % 2 == 0)
                  Xover(one, mem);
            else
                  one = mem;
            }
      }
}
/*************************************************************/
/* Crossover: performs crossover of the two selected parents. */
/*************************************************************/
```

```
void Xover(int one, int two)
{
int i;
int point; /* crossover point */

/* select crossover point */
if(NVARS > 1)
   {
   if(NVARS == 2)
         point = 1;
   else
         point = (rand() % (NVARS - 1)) + 1;

   for (i = 0; i < point; i++)
        swap(&population[one].gene[i], &population[two].gene[i]);

   }
}

/*************************************************************/
/* Swap: A swap procedure that helps in swapping 2 variables */
/*************************************************************/

void swap(int *x, int *y)
{
int temp;

temp = *x;
*x = *y;
*y = temp;

}

/*************************************************************/
/* Mutation: Random uniform mutation. A variable selected for */
/* mutation is replaced by a random value between 1 and m     */
/*************************************************************/

void mutate(void)
{
int i, j;
int lbound, hbound;
double x;

for (i = 0; i < POPSIZE; i++)
      for (j = 0; j < NVARS; j++)
            {
            x = rand()%1000/1000.0;
            if (x < PMUTATION)
                  {
                  /* find the bounds on the variable to be mutated */
                  lbound = population[i].lower[j];
                  hbound = population[i].upper[j];
                  population[i].gene[j] = randval(lbound, hbound);
```

```
                }
            }
}

/*****************************************************************/
/* Main function: follows as it is written                       */
/*****************************************************************/

void main(void)
{
int i, i0, i1;
int WT_RANK[n_g]; /* rank of WT*/
int j[n_g][k];  /* index of affordable points */

double WT[n_g]; /* the smallest ave.wtd.trace for n_g random starts*/

srand(seed);

for (i = 0; i < n_g; i++){
/* Get the Data and Indices */
  get_dat (Dat, Ind);
  generation = 0;
  initialize();
  evaluate();
  keep_the_best();
  while(generation<MAXGENS){
    generation++;
    Select();
    crossover();
    mutate();
    evaluate();
    elitist();
  }
  for (i0 = 0; i0 < k; i0++)
    j[i][i0] = population[POPSIZE].gene[i0];
  WT[i] = 1/population[POPSIZE].fitness;
}
rank (WT, n_g, 'a', WT_RANK);
printf("Genetic Algorithm With %d Random Starts\n", n_g);
printf("m = %d k= %d\n", m, k);
printf("POPSIZE = %d MAXGENS = %d PXOVER = %.2f PMUTATION = %.2f\n",
       POPSIZE, MAXGENS, PXOVER, PMUTATION);

for (i0 = 0; i0 < n_g; i0++) {
  printf("Points:\t");
  for (i1 = 0; i1 < k; i1++) printf("%d\t", j[WT_RANK[i0]][i1]+1);
  printf("\nSides\t");
  for (i1 = 0; i1 < k; i1++) printf("%d\t", Ind[j[WT_RANK[i0]][i1]]);
  printf("\nDat:\t");
  for (i1 = 0; i1 < k; i1++) printf("%.1f\t", Dat[j[WT_RANK[i0]][i1]]);
  printf("%f\n", WT[WT_RANK[i0]]);}
printf("\n");
}

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

B.8. THE PROGRAM "EZ.s"

This S-plus program is for plotting $E(Z_n)$ versus n as discussed in Section 3.4.5. This is the program for Triangle Shape, 0° Orientation, m = 20, k = 4. For other shapes/orientations, one can change the data sets in this program to be other data sets obtained from the ouput of ONE and GA.

```
#-----------------------------------------------------------------------
n _ 10
#For One at a Time (Tri0 -- m = 20, k = 4, 400K Awt)
one _ matrix (scan("/home/gaby/Diss/Tri0/o20.4.t0.dat", multi.line=T),
            ncol=9, byrow=T)[,9]
one _ one[one!=1.e6]    # To eliminate the "(7) Not Invertible" Frequency
one _ as.factor(one)    # make it to be character
one.freq _ table(one)   # frequency table
one.wtrace _ as.numeric(names(one.freq)) # make it numeric
one.rel _ one.freq/sum(one.freq) # relative frequency
one.cum _ cumsum(one.rel) # cumulative frequency
one.tab _ cbind(one.freq, one.rel, one.cum) # combine to be 3 columns

one.k _ nrow(one.tab)
one.n _ n    # No of EZ
one.Q _ matrix (NA, one.k, one.n)
one.Q[,1] _ one.rel

for (j in 2:one.n){
  one.Q[1,j] _ 1 - (1-one.cum[1])^(2*j)
  #one.Q[1,j] _ 1 - (1-one.cum[1])^j
  for (i in 2:one.k){
    one.Q[i,j] _ (1 - one.cum[i-1])^(2*j) - (1 - one.cum[i])^(2*j)}}
    #one.Q[i,j] _ (1 - one.cum[i-1])^j - (1 - one.cum[i])^j}}

one.EZ _ one.wtrace %*% one.Q
cat("One.EZ\n"); print(one.EZ, fill=T)
#-----------------------------------------------------------------------
#### For GA (Tri0 -- m = 20, k = 4, 400K Awt)
ga _ matrix (scan("/home/gaby/Diss/Tri0/g20.4.t0.dat", multi.line=T),
           ncol=9, byrow=T)[,9]
ga _ ga[ga!=1.e6]    # To eliminate the "(7) Not Invertible" Frequency
ga _ as.factor(ga)    # make it to be character
ga.freq _ table(ga)   # frequency table
ga.wtrace _ as.numeric(names(ga.freq)) # make it numeric
ga.rel _ ga.freq/sum(ga.freq) # relative frequency
ga.cum _ cumsum(ga.rel) # cumulative frequency
ga.tab _ cbind(ga.freq, ga.rel, ga.cum) # combine to be 3 columns

ga.k _ nrow(ga.tab)
```

```
ga.n _ n    # No of EZ
ga.Q _ matrix (NA, ga.k, ga.n)
ga.Q[,1] _ ga.rel

for (j in 2:ga.n){
  ga.Q[1,j] _ 1 - (1-ga.cum[1])^j
  for (i in 2:ga.k){
    ga.Q[i,j] _ (1 - ga.cum[i-1])^j - (1 - ga.cum[i])^j}}

ga.EZ _ ga.wtrace %*% ga.Q
cat("Ga.EZ\n"); print(ga.EZ, fill=T)


#plot one.E(Z) & ga.E(Z) table, Z = Min(Y1, ... , Yn)
#motif()
plot(1:n, ylim=range(one.EZ, ga.EZ), xlab = "Computation Resource(GA)",
     ylab="EZ", lty=1, type='n')
lines(1:n, one.EZ, lty=1)
lines(1:n, ga.EZ, lty=2)
legend(8, .52, c("One", "GA"), lty=1:2)
title("Tri0, m=20 and k=4", cex=0.5)
```

B.9. THE PROGRAM "der.s"

Using this Splus program, the derivative functions referred to in Appendix B.4 at the places marked "`/* dfi/d(u,v,t) */`" can be obtained. Before one runs this Splus program, the necessary library needs to be attached. Once it is attached, the next time one runs the program, it does not need to be attached again.

```
#assign(where = 0., "lib.loc", "/home/stat/bin/axp/splus/VR5.3")
#library (MASS, first=T)

dfn <- deriv(expr = f ~ (Y-v-q1*cos(t))*(cos(t)-
                         p1*sin(t))/(sin(t)+p1*cos(t))
                         +u-q1*sin(t),
       namevec=c("u","v","t"),
       function.arg=function(u,v,t, X)NULL)
```

APPENDIX C. COMPUTER OUTPUTS

Presented here are several examples of output files from one of the 3 main programs (exhaust.c, one.c, ga.c). Explanations of what the output files tell us will be given. For the purpose of example, Triangle Shape, 0° Orientation for m = 20 and k = 4, and m = 20 and k = 8 cases are presented.

C.1. THE OUTPUT FILE "tr20.4.t0"

The author recommends that any user who wants to use these program(s) name output file(s) according to some consistent convention. As an example, the author saved the output file from the program "exhaust.c" with the name "tr20.4.t0" where tr refers to triangle exhaustive computation results, 20 refers to m, 4 refers to k, and t0 refers to Triangle Shape and 0° Orientation. The following output gives the 10 Best/Minimum AWTs from Exhaustive Computation. After the 2 title lines, each 2 rows represent a design (ordered from the best/smallest AWT to the tenth best AWT). On the row named "Points:" it shows that for k = 4, the best choice for the first probe path is number 6, for the second probe path the best choice is path number 11, and for the third and fourth paths the best choices are number 20 where the path numbering is as in Figure 3.1. The "Sides" refers to which switch (or derivative functions as in the program "der.h") are used and the last number on that second row gives the AWT computed for that design (i.e., the design "6-11-20-20"). Finally, the number on the bottom row of the output file (i.e., the number "4.285u") is the time (in seconds) needed to run the program.

```
Exausted Search
m=20 k=4
Points:     6     11     20     20
Sides       2     4      6      6      0.438142
Points:     6     11     19     20
Sides       2     4      6      6      0.484586
Points:     6     11     13     20
Sides       2     4      5      6      0.485726
Points:     6     11     12     20
Sides       2     4      4      6      0.519372
Points:     1     6      11     20
Sides       1     2      4      6      0.532119
Points:     6     11     18     20
Sides       2     4      6      6      0.532949
Points:     6     11     11     20
Sides       2     4      4      6      0.550391
Points:     6     10     11     20
Sides       2     3      4      6      0.563399
Points:     6     11     13     19
Sides       2     4      5      6      0.563458
Points:     6     11     14     20
Sides       2     4      5      6      0.565163

4.285u 0.061s 0:04.84 89.6% 0+6k 3+4io 0pf+0w
```

C.2. THE OUTPUT FILE "tr20.8.t0"

The following is the output file for the same object and parameters described above, except the design size is changed to k = 8.

```
Exausted Search
m=20 k=8
Points:     6     11     11     11     20     20     20     20
Sides 2     4     4      4      6      6      6      6      0.193954
Points:     6     11     11     12     20     20     20     20
Sides 2     4     4      4      6      6      6      6      0.200564
Points:     6     11     11     11     19     20     20     20
Sides 2     4     4      4      6      6      6      6      0.204455
Points:     6     11     11     11     13     20     20     20
Sides 2     4     4      4      5      6      6      6      0.204837
Points:     6     11     11     20     20     20     20     20
Sides 2     4     4      6      6      6      6      6      0.206664
Points:     6     11     11     11     12     20     20     20
Sides 2     4     4      4      4      6      6      6      0.206823
Points:     6     11     11     12     19     20     20     20
Sides 2     4     4      4      6      6      6      6      0.210351
Points:     6     11     11     12     13     20     20     20
Sides 2     4     4      4      5      6      6      6      0.211328
```

```
Points:     6     11     11     15     20     20     20     20
Sides 2     4     4      5      6      6      6      6      0.211531
Points:     6     11     11     11     11     20     20     20
Sides 2     4     4      4      4      6      6      6      0.212662

1941.798u 6.089s 1:25:01.18 38.1% 0+957k 19+5io 0pf+0w
```

APPENDIX D. FIGURES PORTRAYING SOME DESIGNS

In this Appendix, figures portraying the best designs found by direct enumeration (via the program "exhaust.c" as in Appendix B) for "small" and "medium" (m, k) combinations are displayed in Sections D.1 through D.7. Sections D.8 through D.23 give figures for best designs found by ONE and GA where no direct enumeration was possible (i.e., for the "large" (m, k) combination).

D.1. TRIANGLE, 45° ORIENTATION

Figure D.1 shows the Triangle, 45° Orientation case for both "small" and "medium" (m, k) combinations, i.e., (m, k) = (20, 4) and (20, 8). In Figure D.1, the symbol "[2X]" represents repetition of the probe path for k = 4, while the symbols "(1X)", "(2X)", "(5X)" represent how many times those probe paths are employed for k = 8. Apparently, as k increases from 4 to 8 points, the best paths are being repeated.

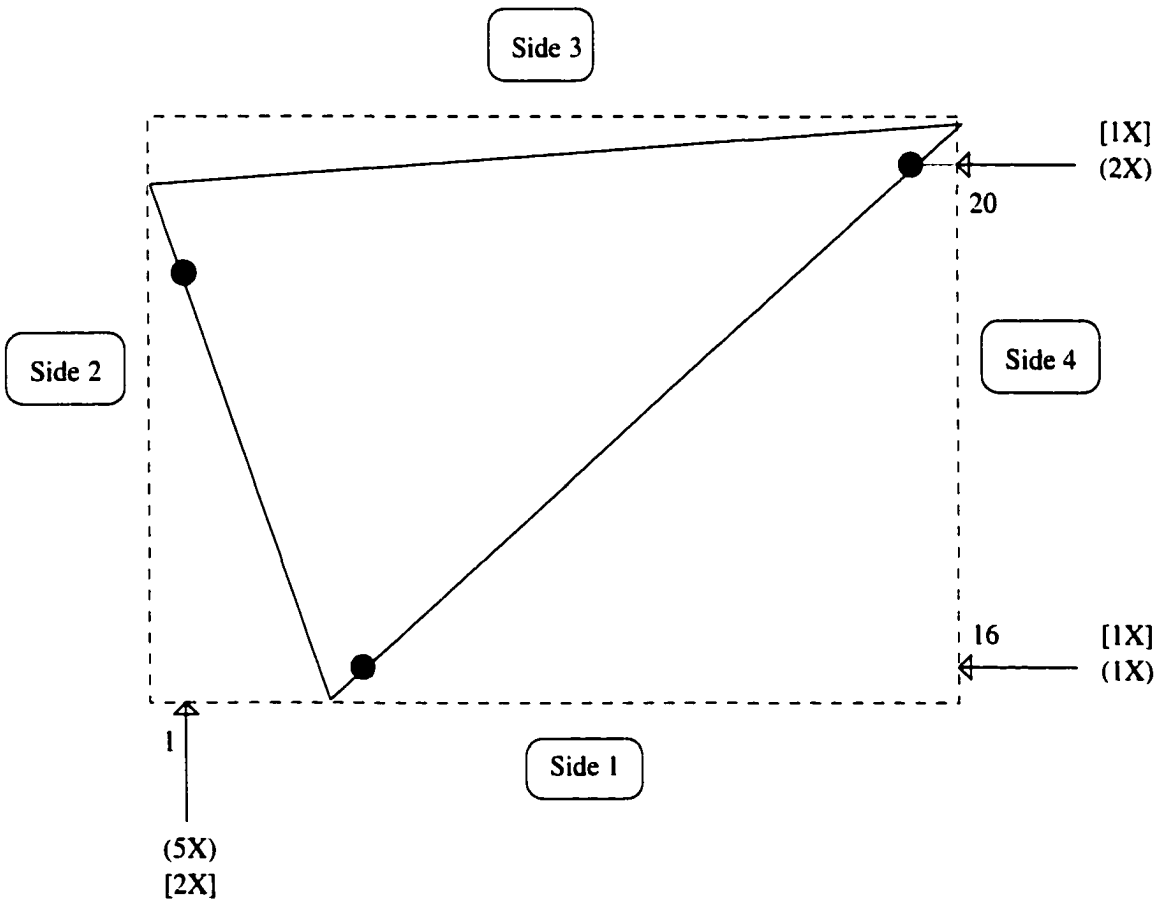

Figure D.1. Best Designs for Triangle Shape, 45° Orientation, and "Small"/"Medium" (m, k) Combinations; Nominal Position ($\underline{\beta} = \underline{0}$) Shown

D.2. RECTANGLE, 0° ORIENTATION

Figure D.2 shows best designs for the Rectangle, 0° Orientation case for both "small" and "medium" (m, k) combinations, i.e., (m, k) = (20, 4) and (20, 8). As before, [ ] indicates repetition of paths for the k = 4 case and ( ) indicates repetition of paths for the k = 8 case.

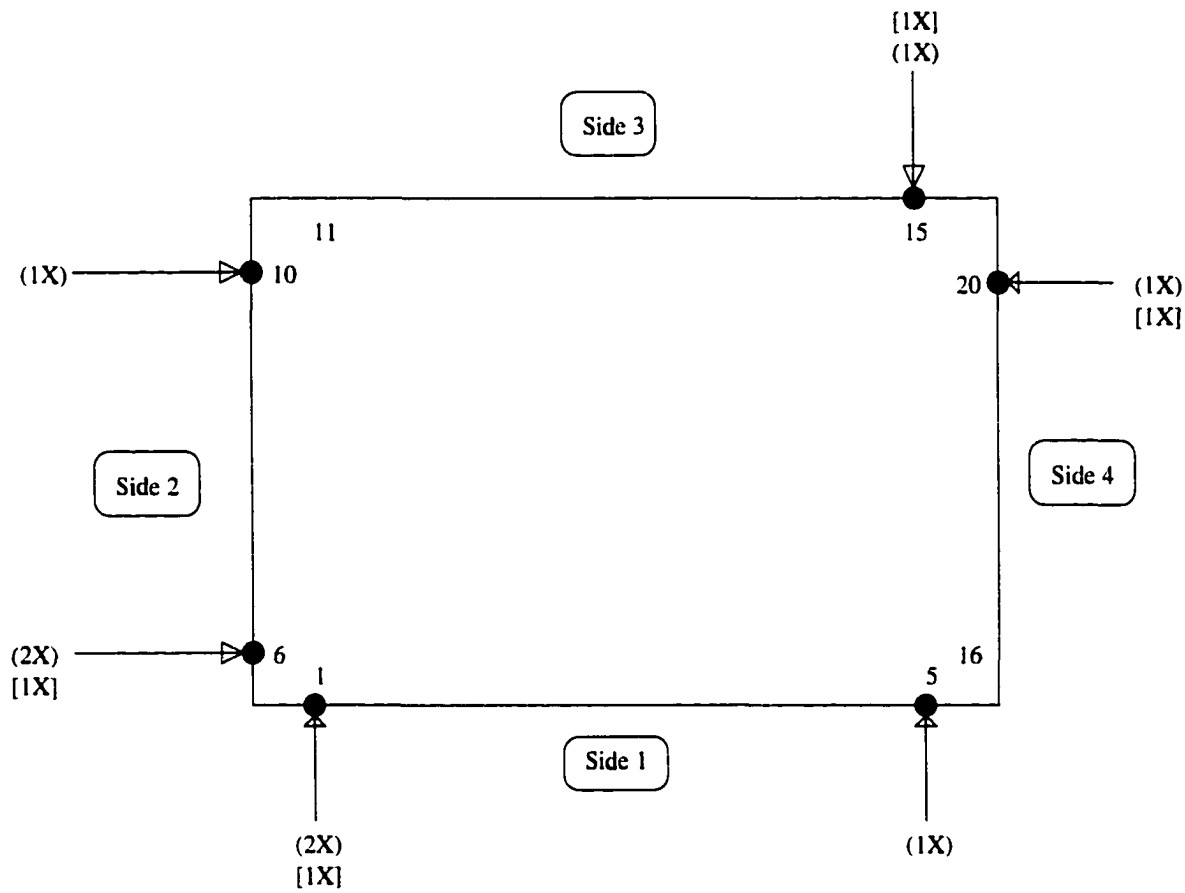

Figure D.2. Best Designs for Rectangle Shape, 0° Orientation, and "Small"/"Medium" (m, k) Combinations; Nominal Position ($\underline{\beta} = \underline{0}$) Shown

D.3. RECTANGLE, 45° ORIENTATION

Figure D.3 shows the Rectangle, 45° Orientation case for both "small" and "medium" (m, k) combinations, i.e., (m, k) = (20, 4) and (20, 8).

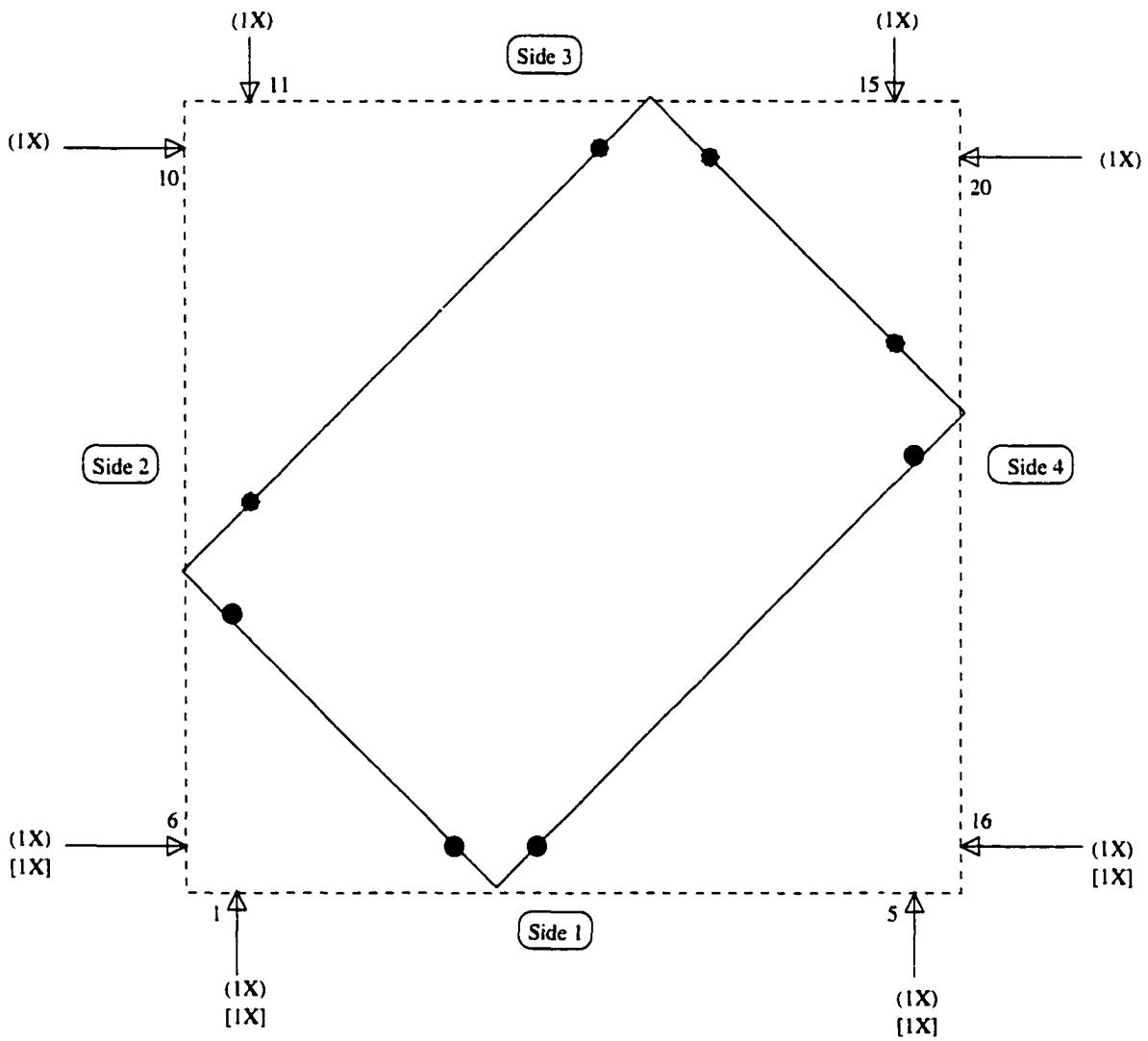

Figure D.3. Best Designs for Rectangle Shape, 45° Orientation, and "Small"/"Medium" (m, k) Combinations; Nominal Position ($\underline{\beta} = \underline{0}$) Shown

D.4. HOOD SHAPE, 0° ORIENTATION

Figure D.4 shows the Hood Shape, 0° Orientation case for both "small" and "medium" (m, k) combinations, i.e., (m, k) = (20, 4) and (20, 8).

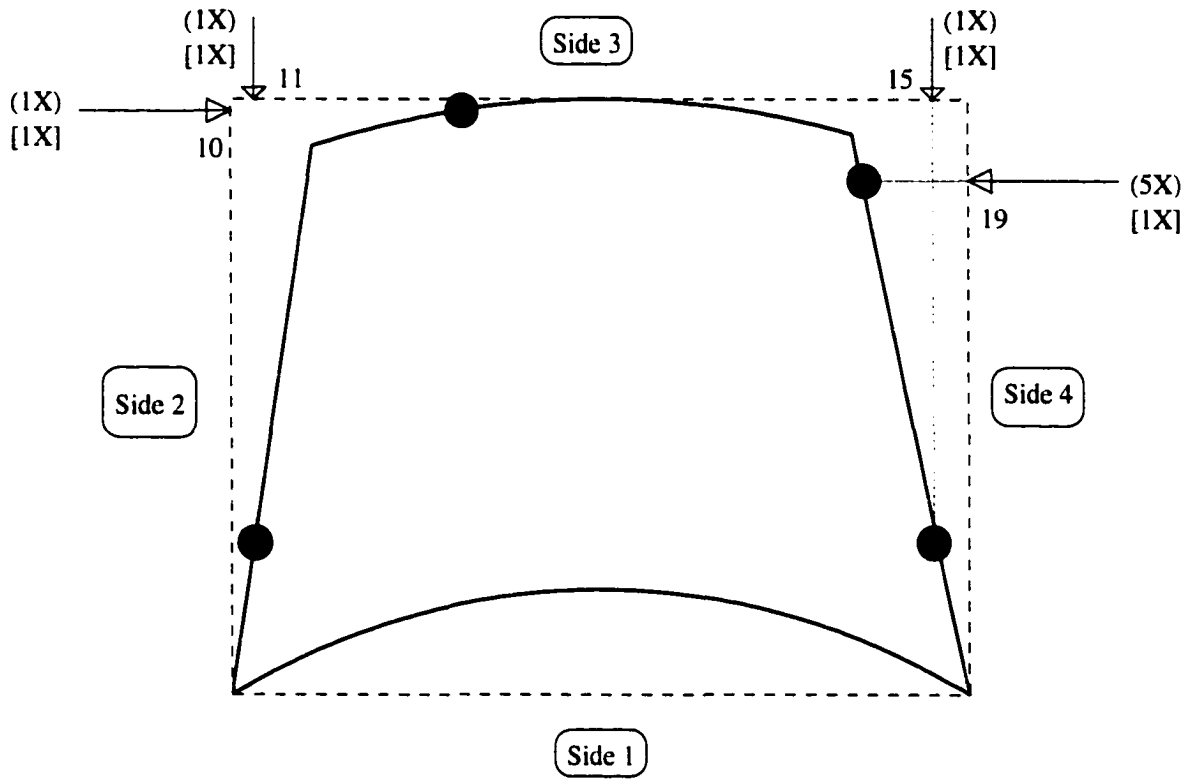

Figure D.4. Best Designs for Hood Shape, 0° Orientation, and "Small"/"Medium" (m, k) Combinations; Nominal Position ($\underline{\beta} = \underline{0}$) Shown

D.5. HOOD SHAPE, 45° ORIENTATION

Figure D.5 shows the Hood Shape, 45° Orientation case for both "small" and "medium" (m, k) combinations, i.e., (m, k) = (20, 4) and (20, 8).

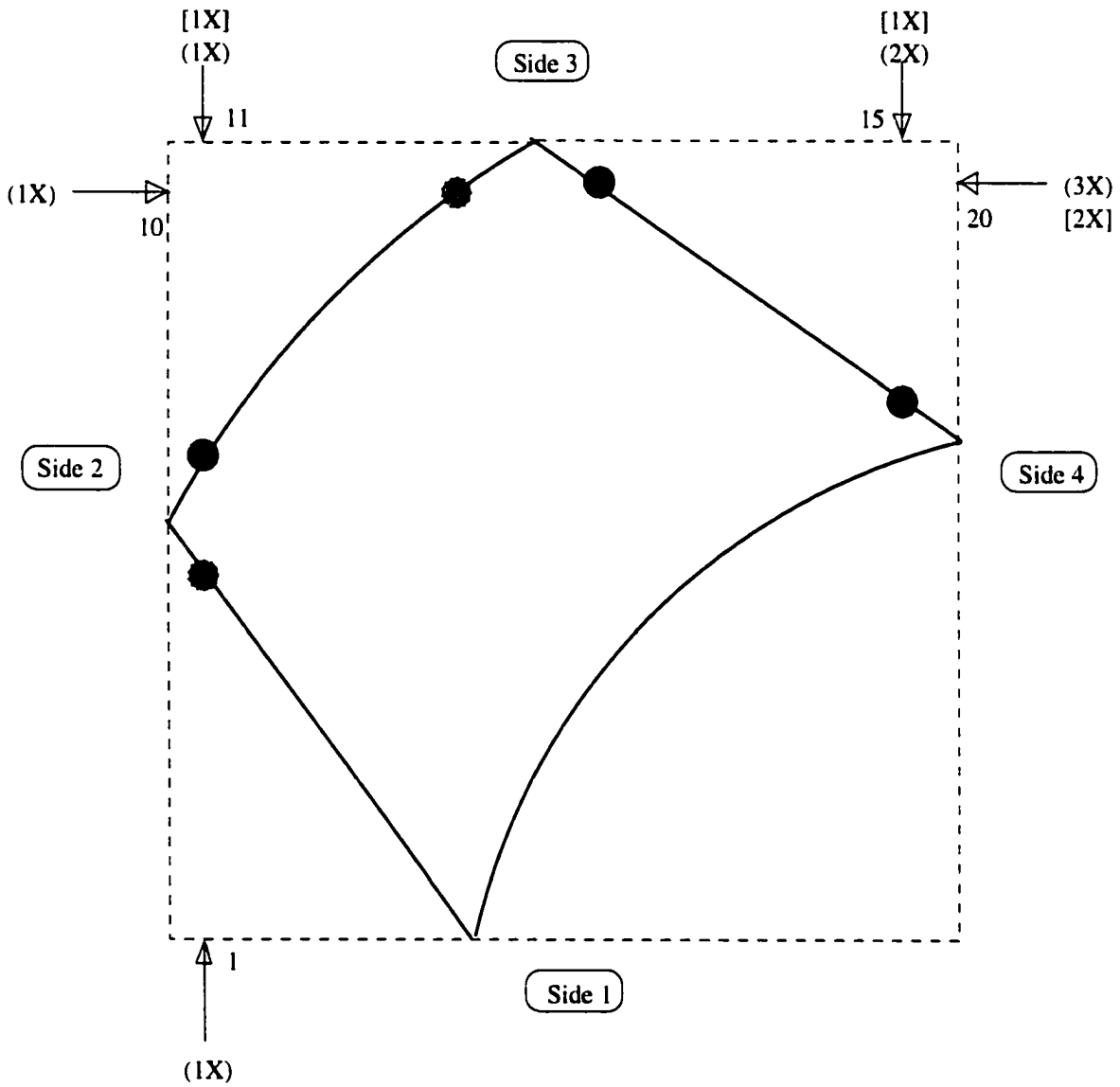

Figure D.5. Best Designs for Hood Shape, 45° Orientation, and "Small"/"Medium" (m, k) Combinations; Nominal Position ($\underline{\beta} = \underline{0}$) Shown

D.6. PARABOLAS SHAPE, 0° ORIENTATION

Figure D.6 shows the Parabolas Shape, 0° Orientation case for both "small" and "medium" (m, k) combinations, i.e., (m, k) = (20, 4) and (20, 8).

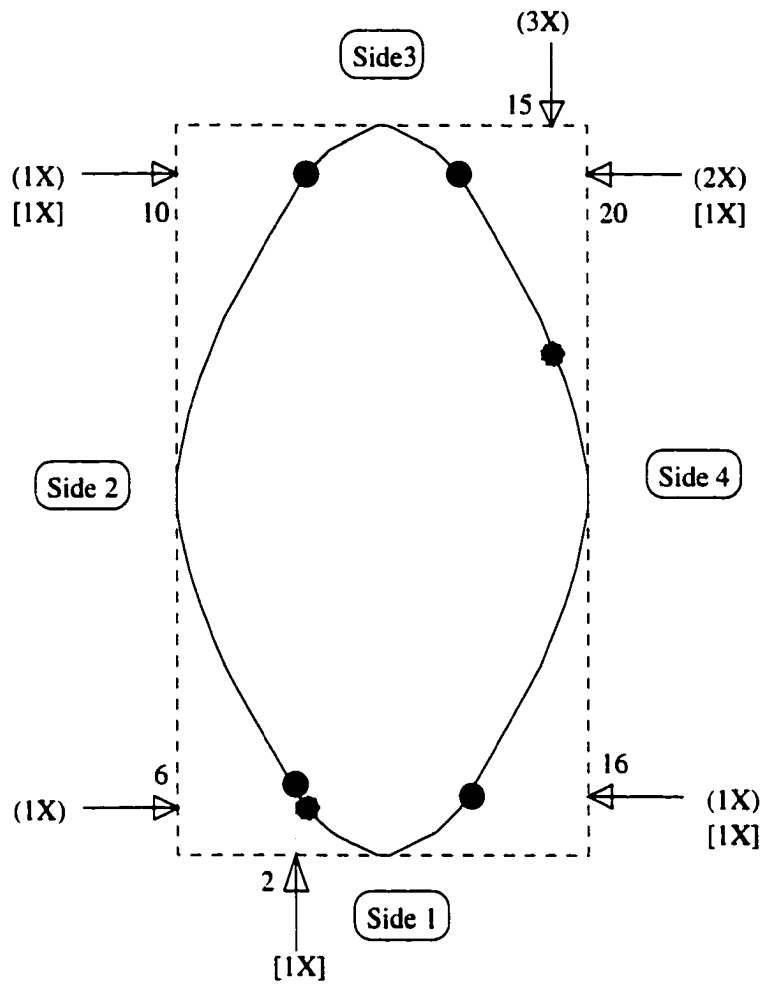

Figure D.6. Best Designs for Parabolas Shape, 0° Orientation, and "Small"/"Medium" (m, k) Combinations; Nominal Position ($\underline{\beta} = \underline{0}$) Shown

D.7. PARABOLAS SHAPE, 45° ORIENTATION

Figure D.7 shows the Parabolas Shape, 45° Orientation case for both "small" and "medium" (m, k) combinations, i.e., (m, k) = (20, 4) and (20, 8).

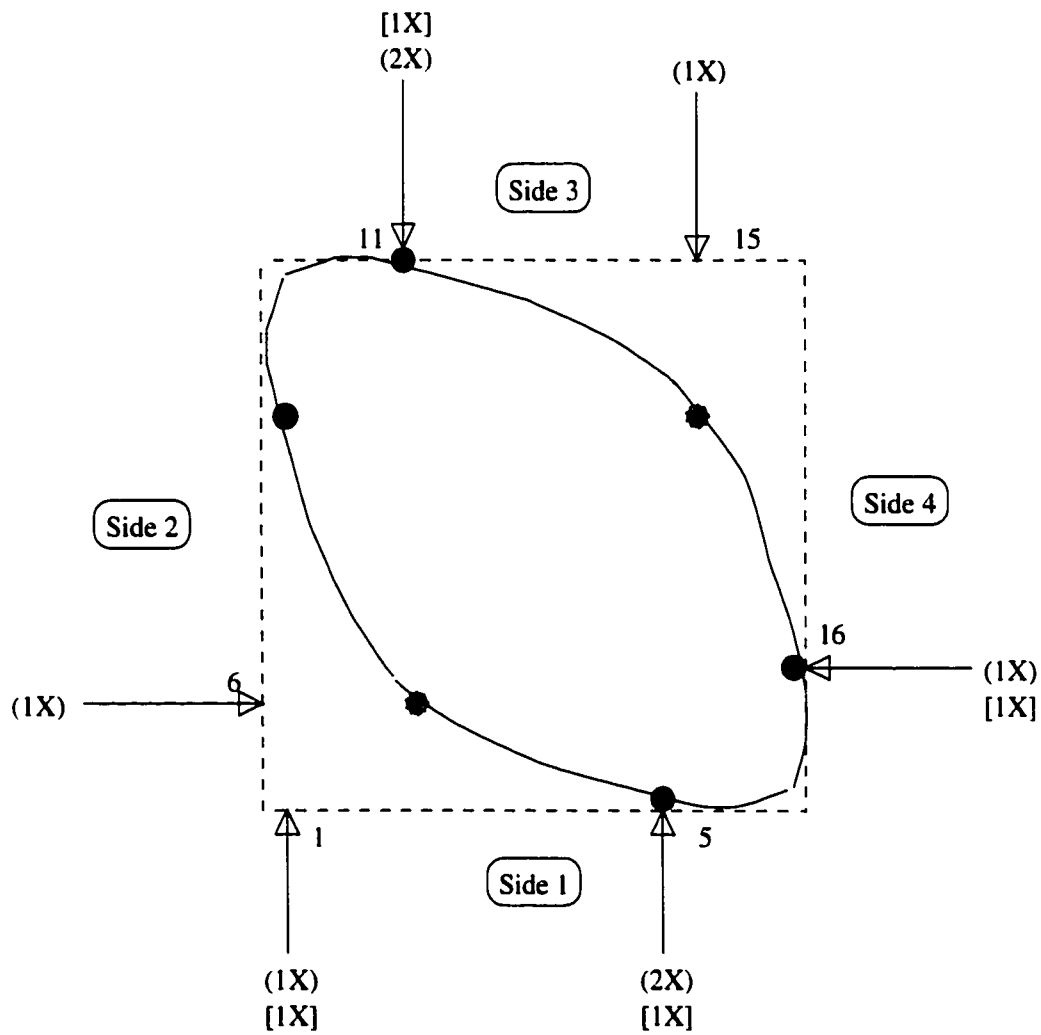

Figure D.7. Best Designs for Parabolas Shape, 45° Orientation, and "Small"/"Medium" (m, k) Combinations; Nominal Position ($\underline{\beta} = \underline{0}$) Shown

D.8. TRIANGLE SHAPE, 0° ORIENTATION

Figure D.8 portrays the best design found by GA for the Triangle Shape, 0° Orientation case for "large" (m, k), i.e., (m, k) = (40, 16) where no direct enumeration was possible.

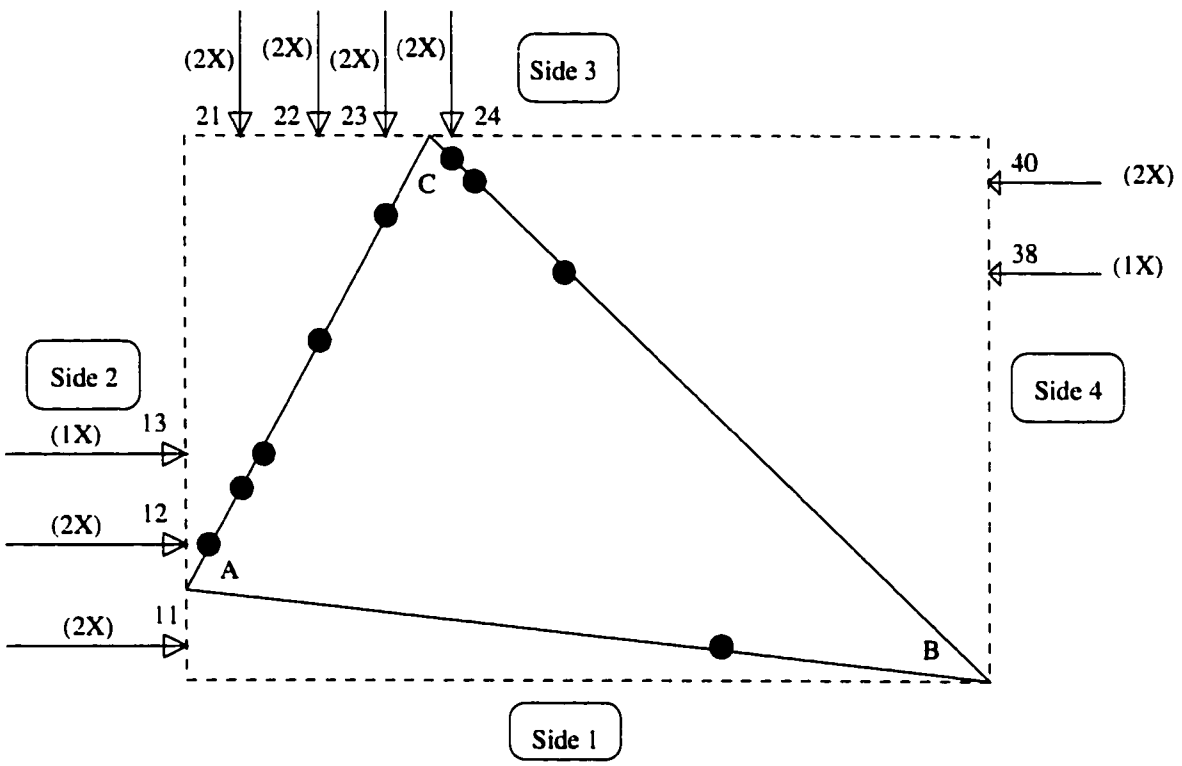

Figure D.8. The Best Design Found by GA for the Triangle Shape, 0° Orientation, and "Large" (m, k) Case; Nominal Position ($\underline{\beta} = \underline{0}$) Shown

D.9. TRIANGLE SHAPE, 0° ORIENTATION

Figure D.9 portrays the best design found by ONE for the Triangle Shape, 0° Orientation case for "large" (m, k), i.e., (m, k) = (40, 16) where no direct enumeration was possible.

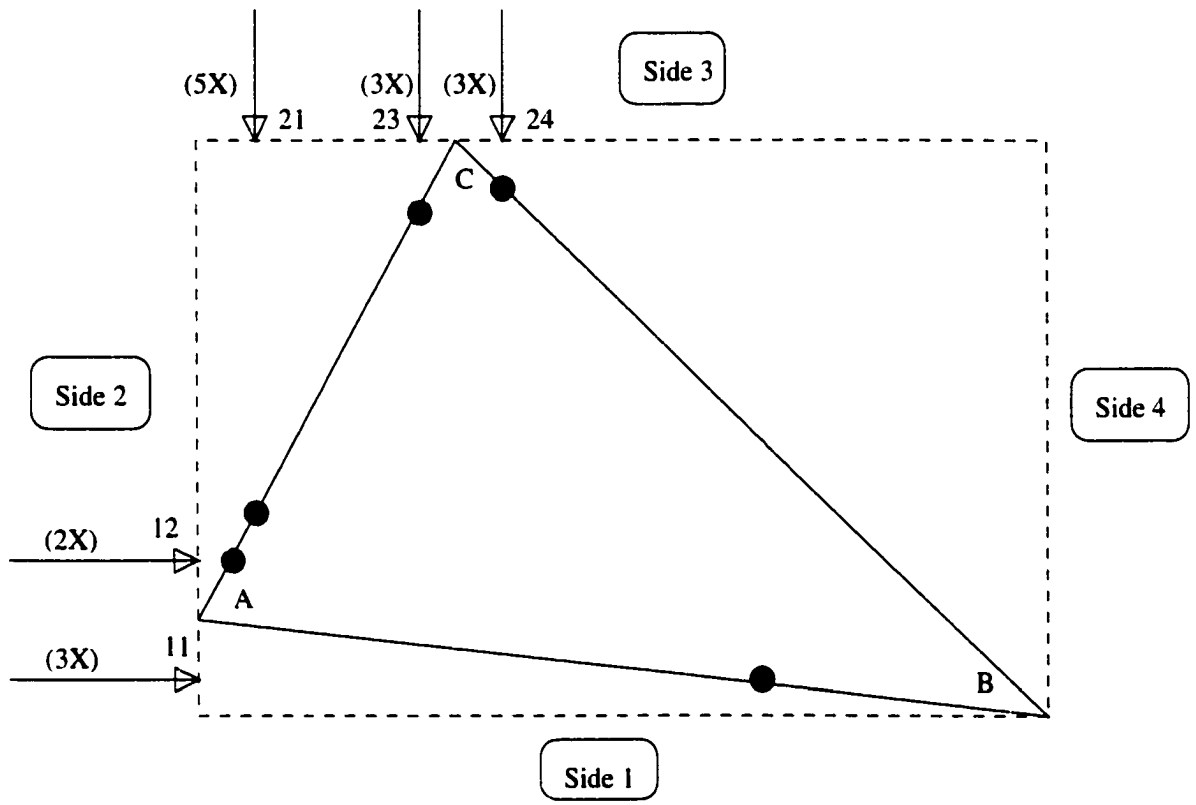

Figure D.9. The Best Design Found by ONE for the Triangle Shape, 0° Orientation, and "Large" (m, k) Case; Nominal Position ($\underline{\beta} = \underline{0}$) Shown

D.10. TRIANGLE SHAPE, 45° ORIENTATION

Figure D.10 portrays the best design found by GA for the Triangle Shape, 45° Orientation case for "large" (m, k), i.e., (m, k) = (40, 16) where no direct enumeration was possible.

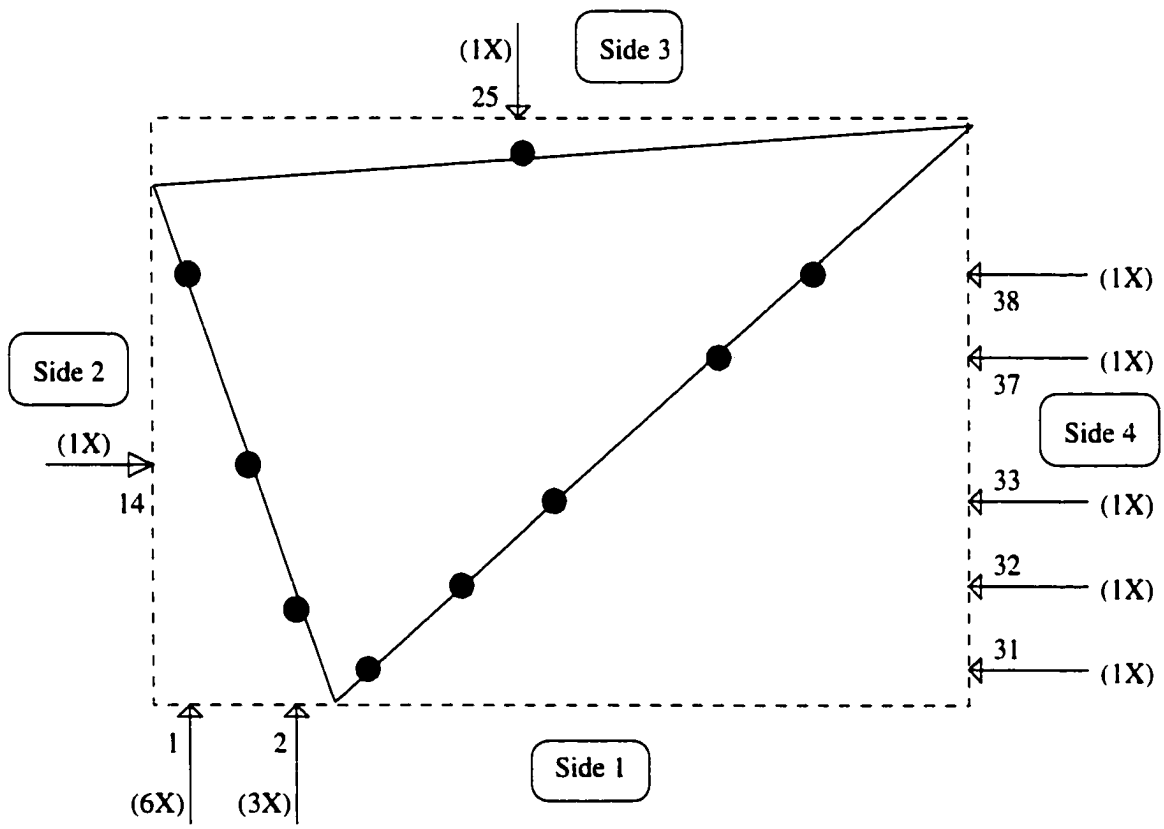

Figure D.10. The Best Design Found by GA for the Triangle Shape, 45° Orientation, and "Large" (m, k) Case; Nominal Position ($\underline{\beta} = \underline{0}$) Shown

D.11. TRIANGLE SHAPE, 45° ORIENTATION

Figure D.11 portrays the best design found by ONE for the Triangle Shape, 45° Orientation case for "large" (m, k), i.e., (m, k) = (40, 16) where no direct enumeration was possible.

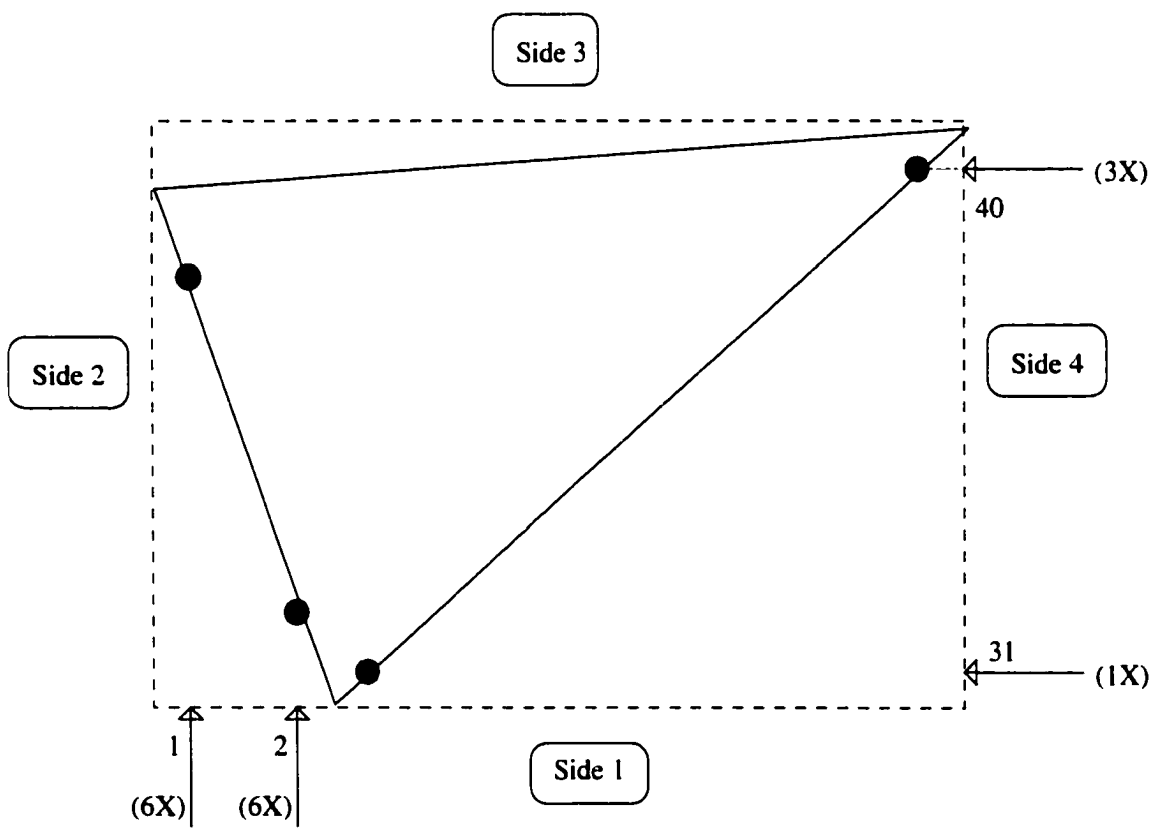

Figure D.11. The Best Design Found by ONE for the Triangle Shape, 45° Orientation, and "Large" (m, k) Case; Nominal Position ($\underline{\beta} = \underline{0}$) Shown

D.12. RECTANGLE SHAPE, 0° ORIENTATION

Figure D.12 portrays the best design found by GA for the Rectangle Shape, 0° Orientation case for "large" (m, k), i.e., (m, k) = (40, 16) where no direct enumeration was possible.

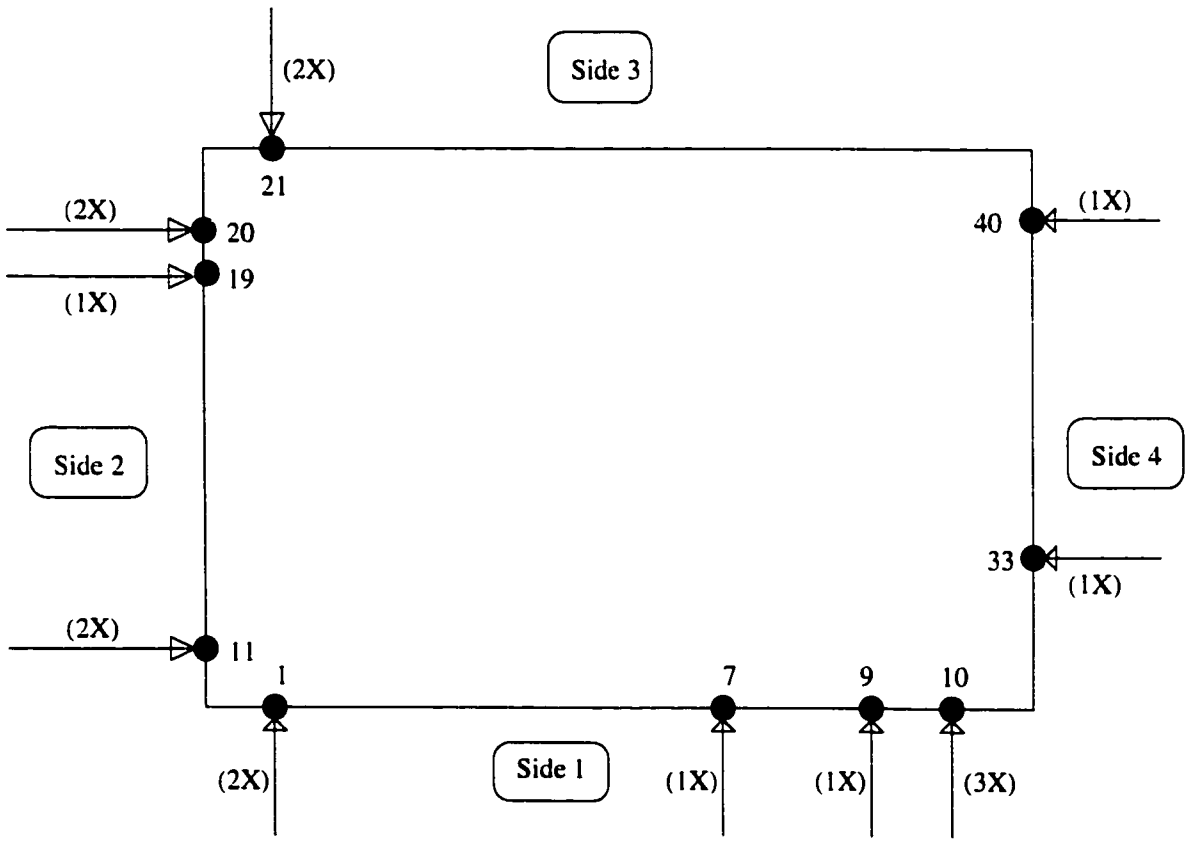

Figure D.12. The Best Design Found by GA for the Rectangle Shape, 0° Orientation, and "Large" (m, k) Case; Nominal Position ($\underline{\beta} = \underline{0}$) Shown

D.13. RECTANGLE SHAPE, 0° ORIENTATION

Figure D.13 portrays the best design found by ONE for the Rectangle Shape, 0° Orientation case for "large" (m, k), i.e., (m, k) = (40, 16) where no direct enumeration was possible.

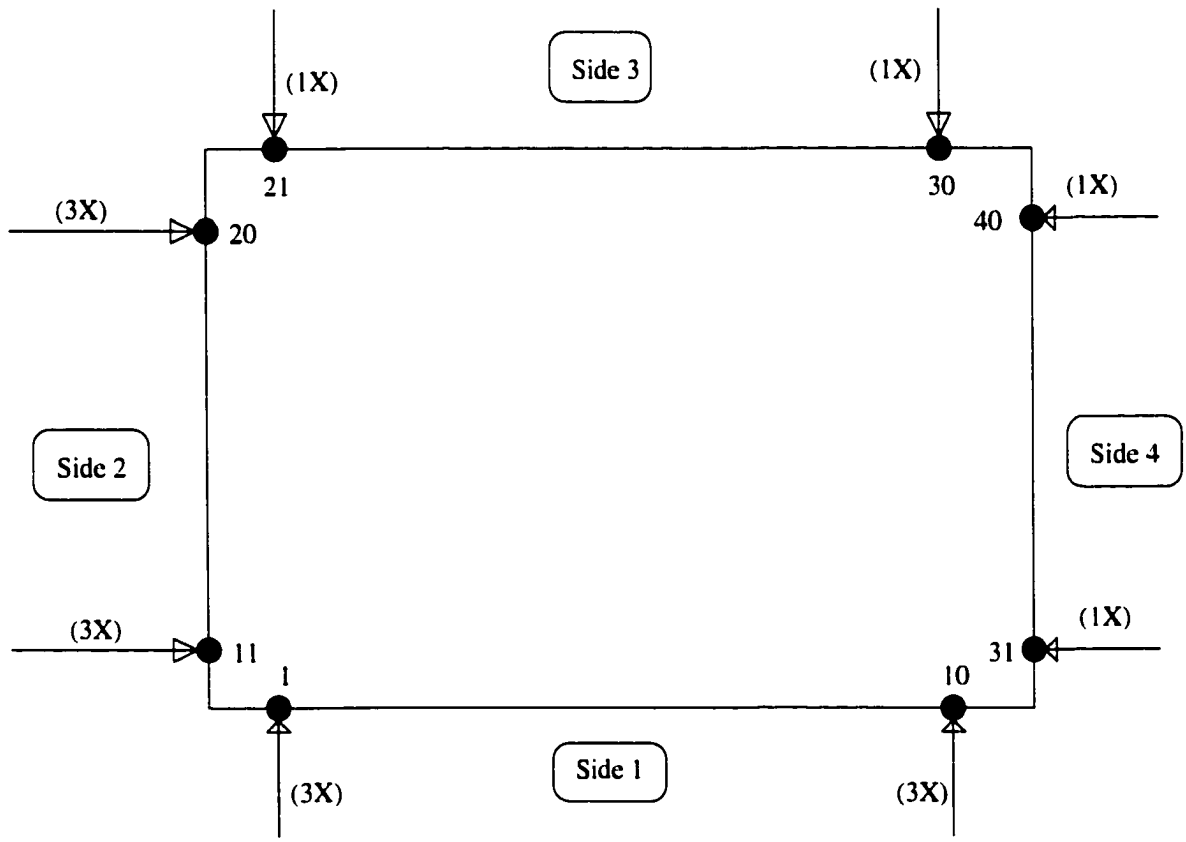

Figure D.13. The Best Design Found by ONE for the Rectangle Shape, 0° Orientation, and "Large" (m, k) Case; Nominal Position ($\underline{\beta} = \underline{0}$) Shown

D.14. RECTANGLE SHAPE, 45° ORIENTATION

Figure D.14 portrays the best design found by GA for the Rectangle Shape, 45° Orientation case for "large" (m, k), i.e., (m, k) = (40, 16) where no direct enumeration was possible.

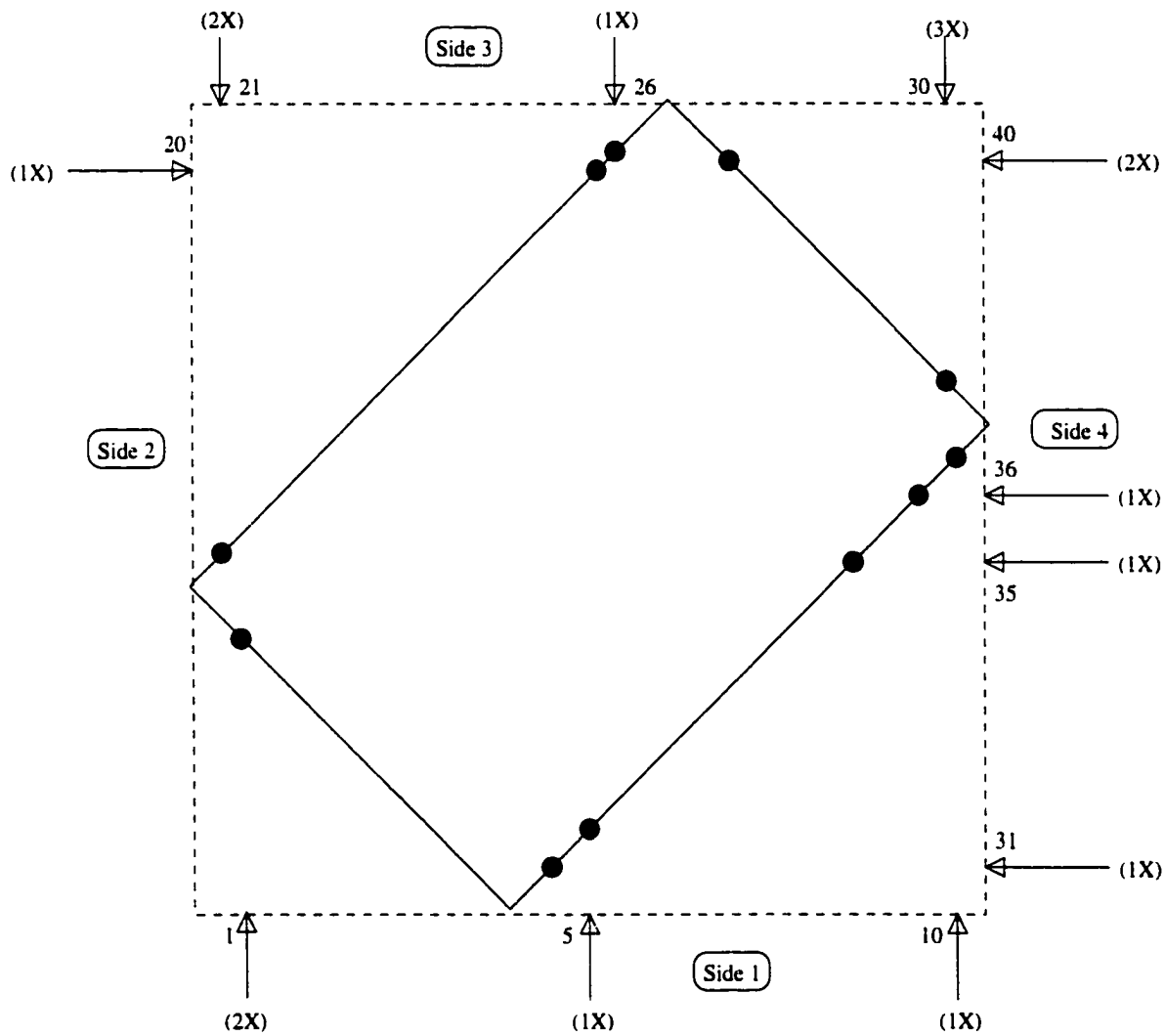

Figure D.14. The Best Design Found by GA for the Rectangle Shape, 45° Orientation, and "Large" (m, k) Case; Nominal Position ($\underline{\beta} = \underline{0}$) Shown

D.15. RECTANGLE SHAPE, 45° ORIENTATION

Figure D.15 portrays the best design found by ONE for the Rectangle Shape, 45° Orientation case for "large" (m, k), i.e., (m, k) = (40, 16) where no direct enumeration was possible.

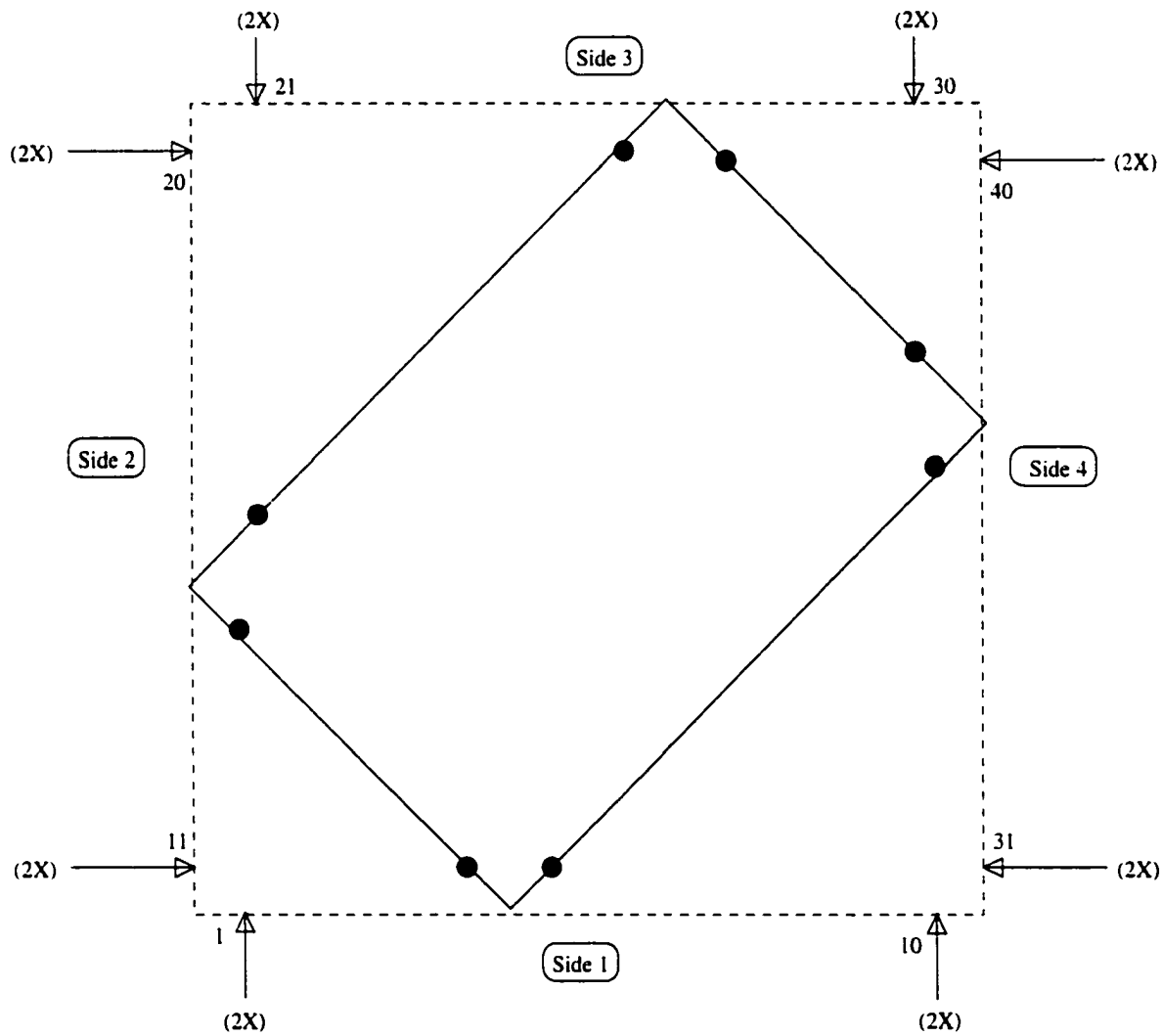

Figure D.15. The Best Design Found by ONE for the Rectangle Shape, 45° Orientation, and "Large" (m, k) Case; Nominal Position ($\underline{\beta} = \underline{0}$) Shown

D.16. HOOD SHAPE, 0° ORIENTATION

Figure D.16 portrays the best design found by GA for the Hood Shape, 0° Orientation case for "large" (m, k), i.e., (m, k) = (40, 16) where no direct enumeration was possible.

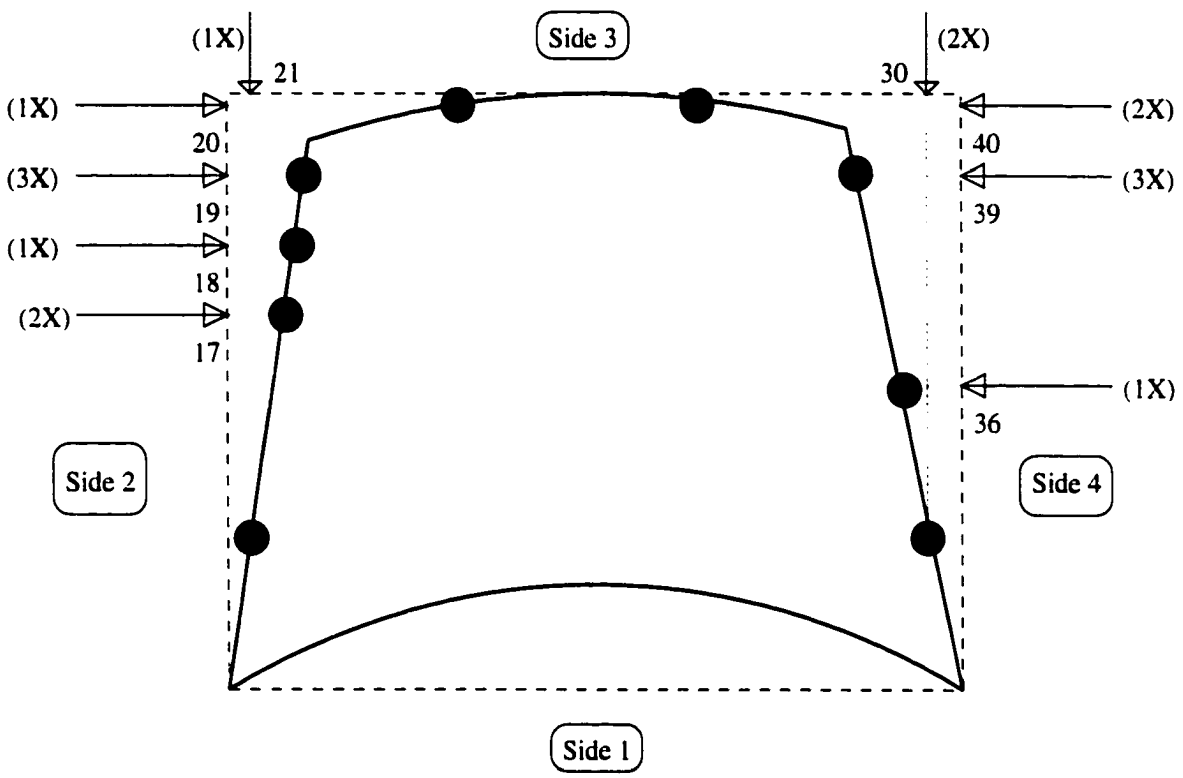

Figure D.16. The Best Design Found by GA for the Hood Shape, 0° Orientation, and "Large" (m, k) Case; Nominal Position ($\underline{\beta} = \underline{0}$) Shown

D.17. HOOD SHAPE, 0° ORIENTATION

Figure D.17 portrays the best design found by ONE for the Hood Shape, 0° Orientation case for "large" (m, k), i.e., (m, k) = (40, 16) where no direct enumeration was possible.

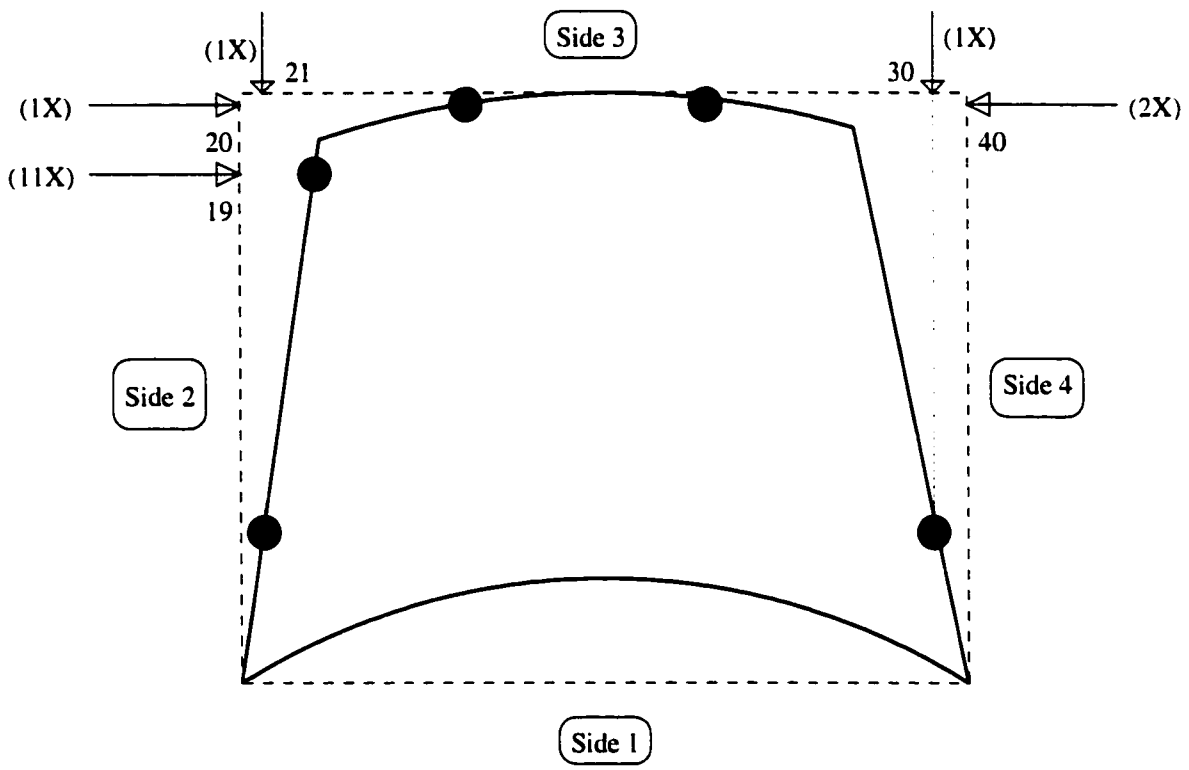

Figure D.17. The Best Design Found by ONE for the Hood Shape, 0° Orientation, and "Large" (m, k) Case; Nominal Position ($\underline{\beta} = \underline{0}$) Shown

D.18. HOOD SHAPE, 45° ORIENTATION

Figure D.18 portrays the best design found by GA for the Hood Shape, 45° Orientation case for "large" (m, k), i.e., (m, k) = (40, 16) where no direct enumeration was possible.

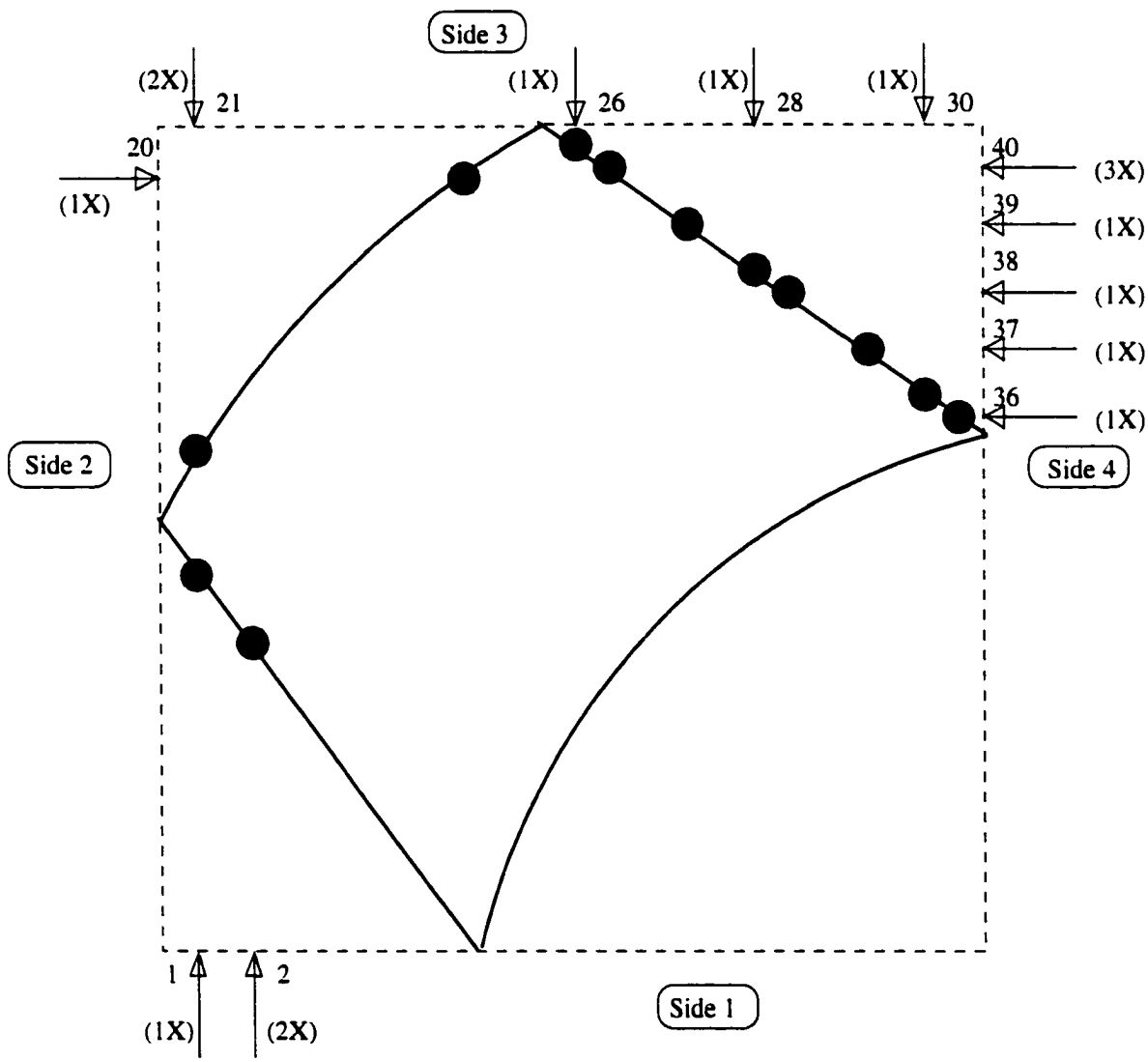

Figure D.18. The Best Design Found by GA for the Hood Shape, 45° Orientation, and "Large" (m, k) Case; Nominal Position ($\underline{\beta} = \underline{0}$) Shown

D.19. HOOD SHAPE, 45° ORIENTATION

Figure D.19 portrays the best design found by ONE for the Hood Shape, 45° Orientation case for "large" (m, k), i.e., (m, k) = (40, 16) where no direct enumeration was possible.

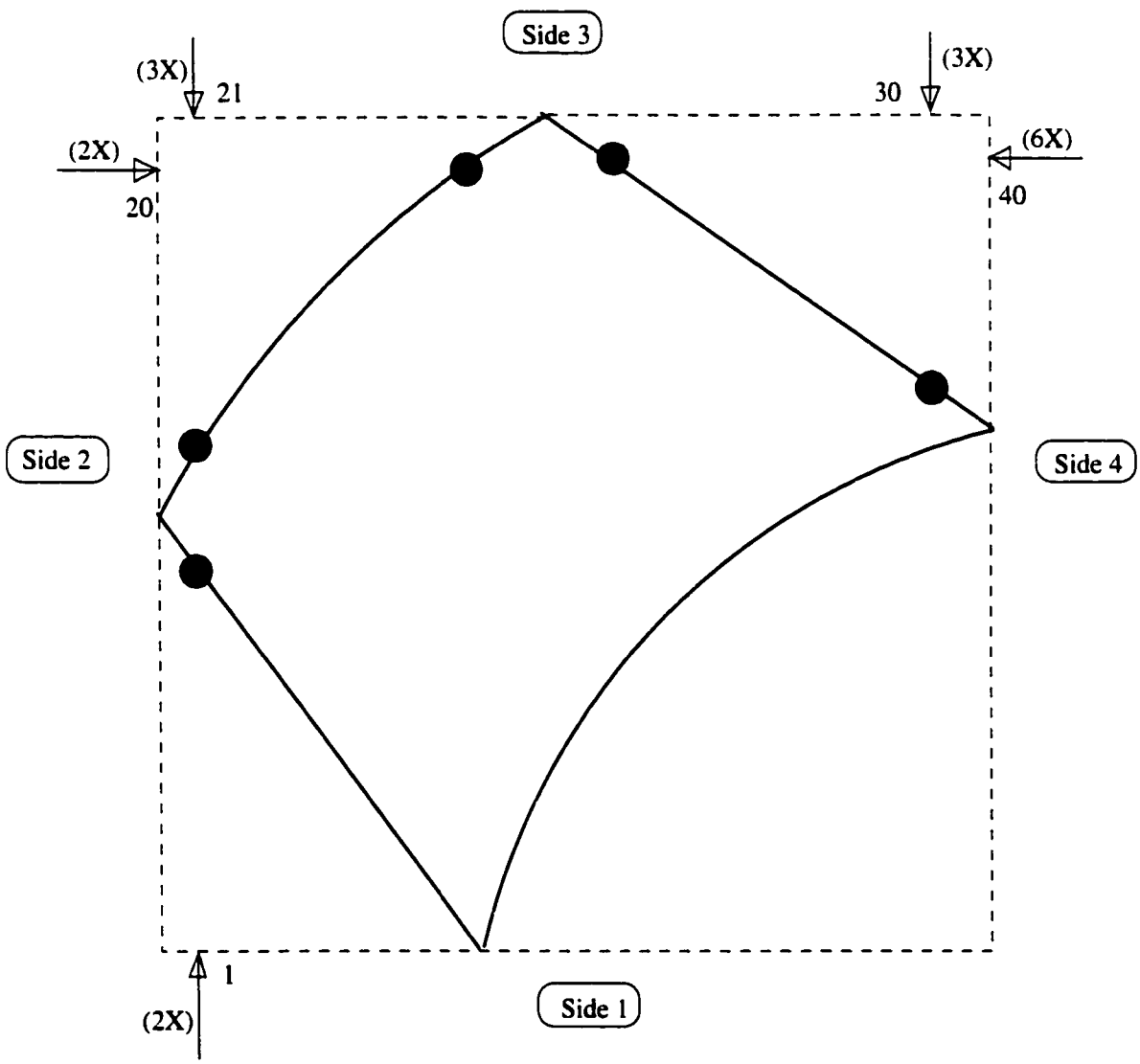

Figure D.19. The Best Design Found by ONE for the Hood Shape, 45° Orientation, and "Large" (m, k) Case; Nominal Position ($\underline{\beta} = \underline{0}$) Shown

D.20. PARABOLAS SHAPE, 0° ORIENTATION

Figure D.20 portrays the best design found by GA for the Parabolas Shape, 0° Orientation case for "large" (m, k), i.e., (m, k) = (40, 16) where no direct enumeration was possible.

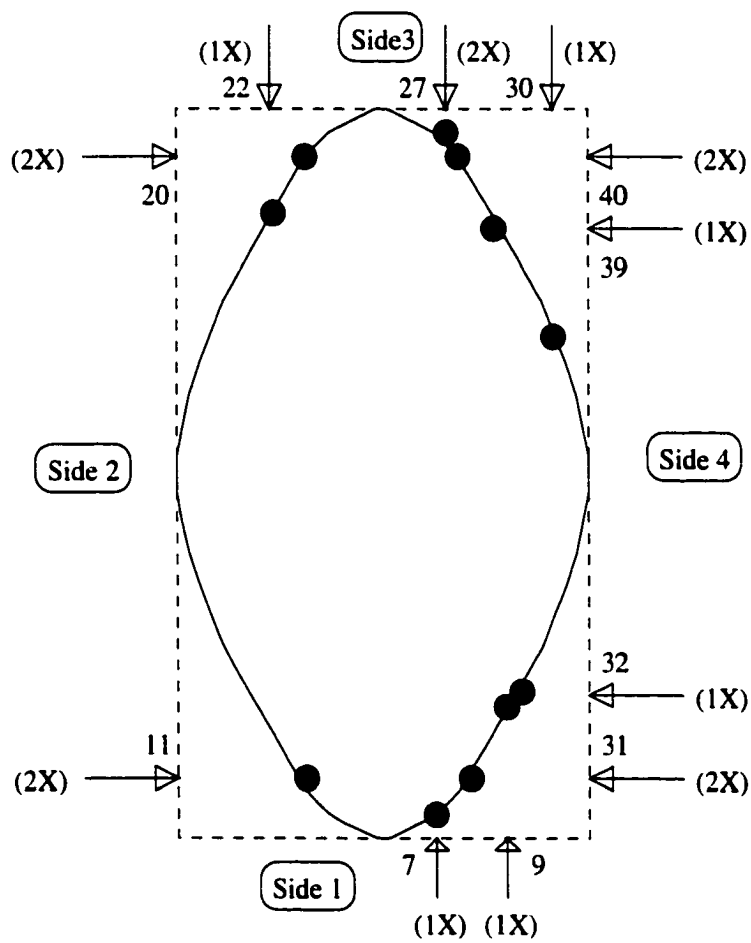

Figure D.20. The Best Design Found by GA for the Parabolas Shape, 0° Orientation, and "Large" (m, k) Case; Nominal Position ($\underline{\beta} = \underline{0}$) Shown

D.21. PARABOLAS SHAPE, 0° ORIENTATION

Figure D.21 portrays the best design found by ONE for the Parabolas Shape, 0° Orientation case for "large" (m, k), i.e., (m, k) = (40, 16) where no direct enumeration was possible.

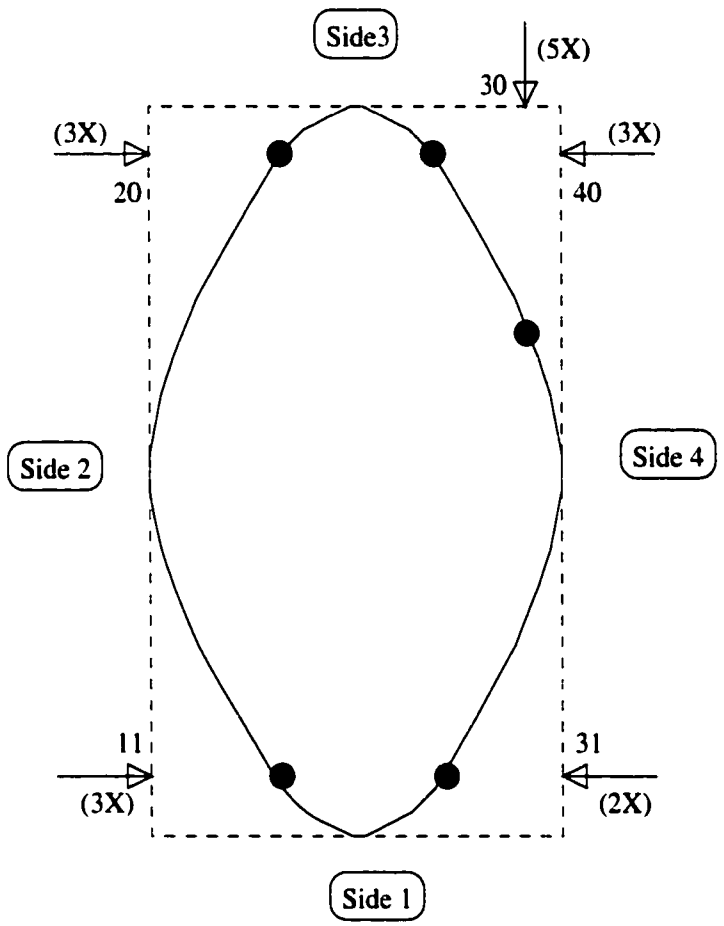

Figure D.21. The Best Design Found by ONE for the Parabolas Shape, 0° Orientation, and "Large" (m, k) Case; Nominal Position ($\underline{\beta} = \underline{0}$) Shown

D.22. PARABOLAS SHAPE, 45° ORIENTATION

Figure D.22 portrays the best design found by GA for the Parabolas Shape, 45° Orientation case for "large" (m, k), i.e., (m, k) = (40, 16) where no direct enumeration was possible.

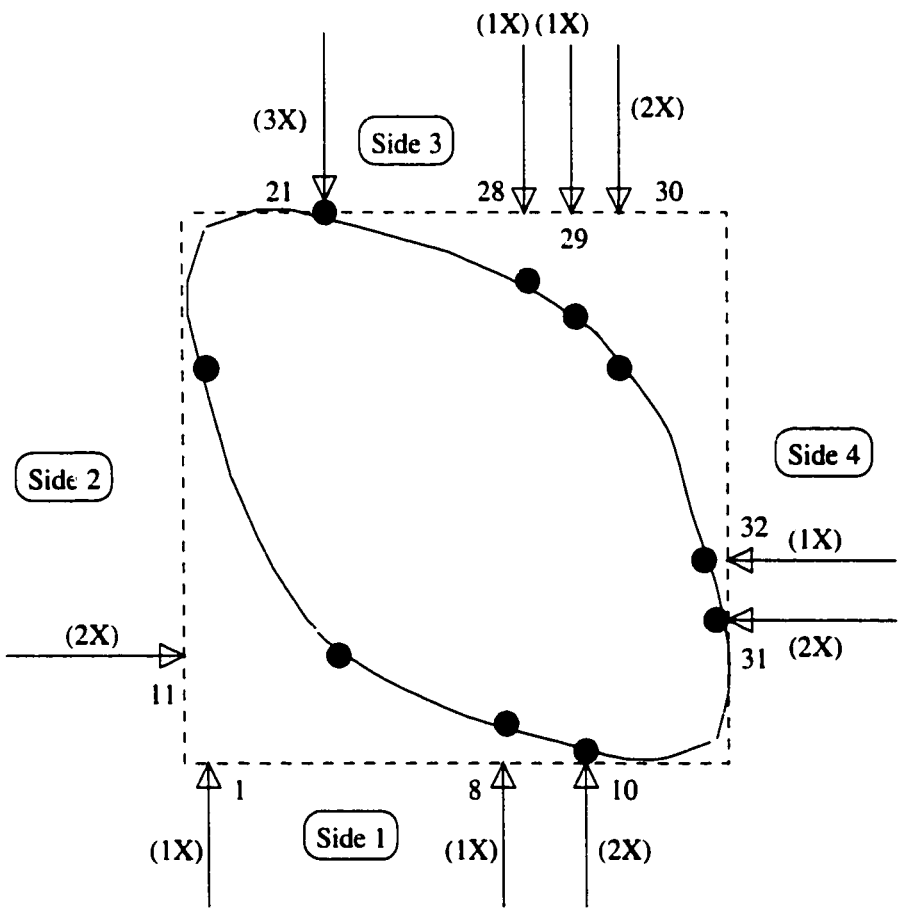

Figure D.22. The Best Design Found by GA for the Parabolas Shape, 45° Orientation, and "Large" (m, k) Case; Nominal Position ($\underline{\beta} = \underline{0}$) Shown

D.23. PARABOLAS SHAPE, 45° ORIENTATION

Figure D.23 portrays the best design found by ONE for the Parabolas Shape, 45° Orientation case for "large" (m, k), i.e., (m, k) = (40, 16) where no direct enumeration was possible.

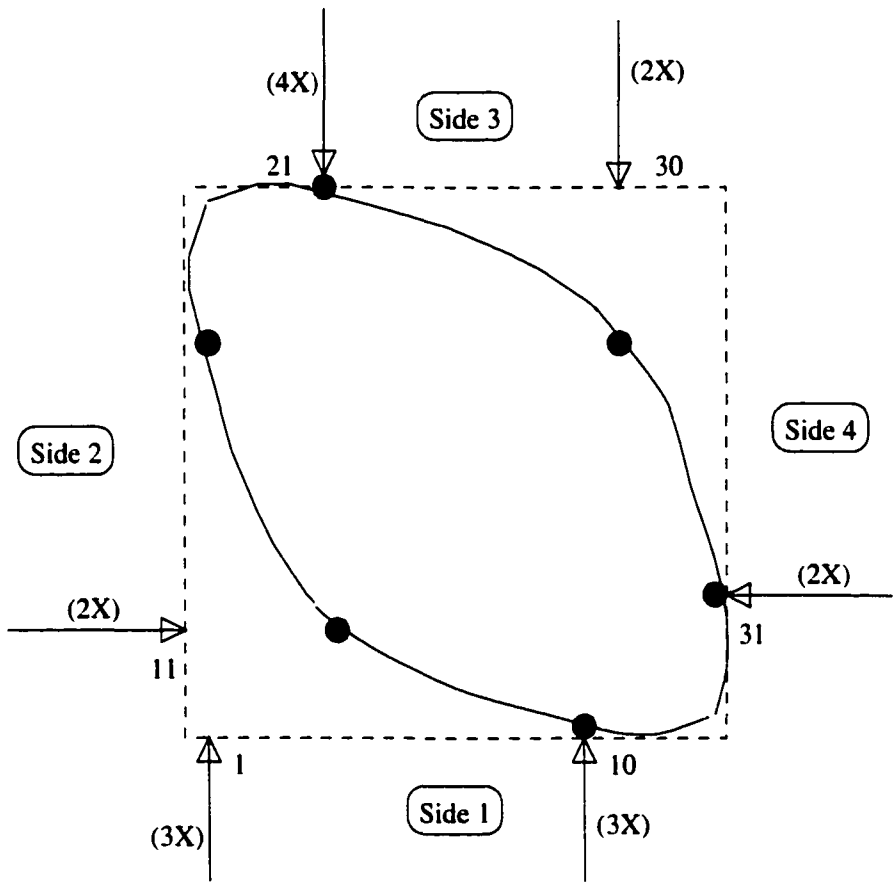

Figure D.23. The Best Design Found by ONE for the Parabolas Shape, 45° Orientation, and "Large" (m, k) Case; Nominal Position ($\underline{\beta} = \underline{0}$) Shown

APPENDIX E. GRAPHS (ONE VERSUS GA)

The following are the graphs of $E(Z_n)$ versus n for both ONE and GA as discussed in Section 3.4.5. Figure E.1 is for various combinations of orientations and (m, k) for the Triangle Shape. Figure E.2 is for the Rectangle Shape. Figure E.3 is for the Hood Shape. Figure E.4 is for the Parabolas Shape.

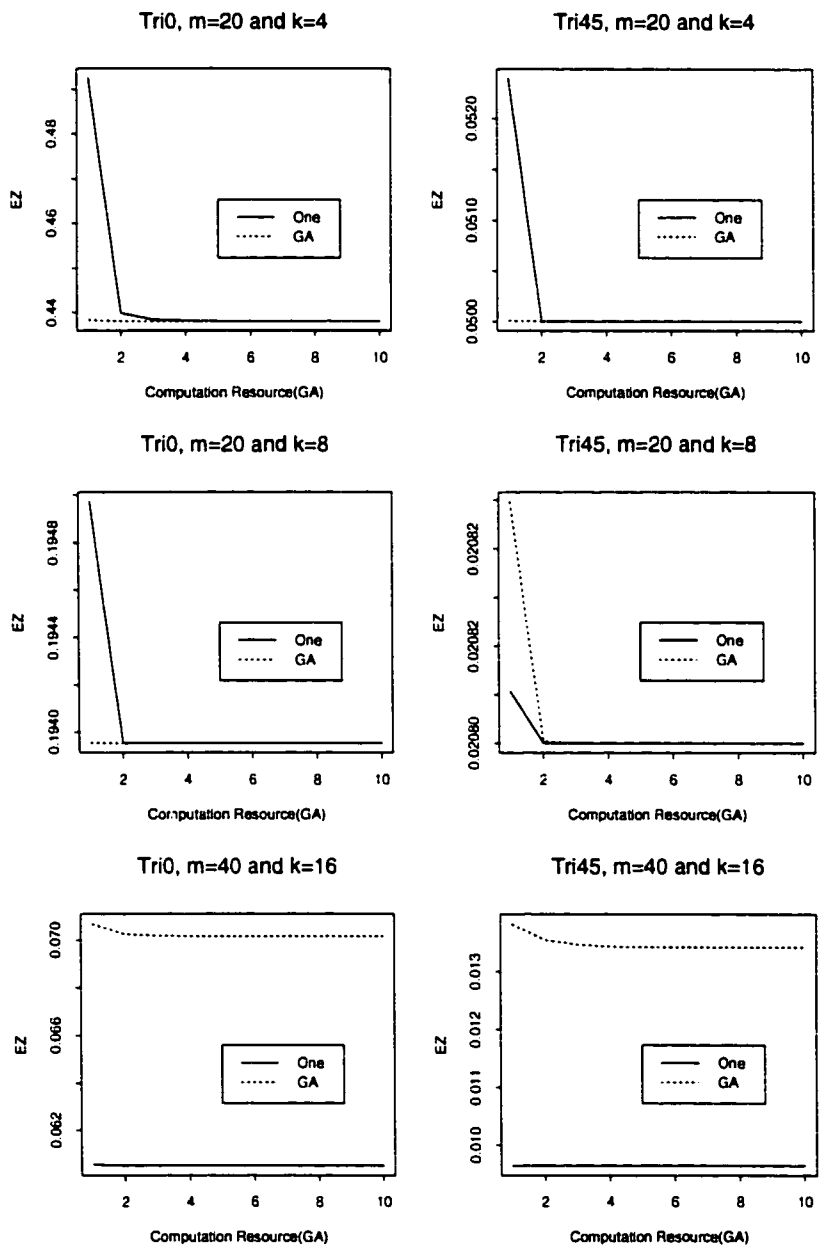

Figure E.1. $E(Z_n)$ versus n (Triangle Shape combinations)

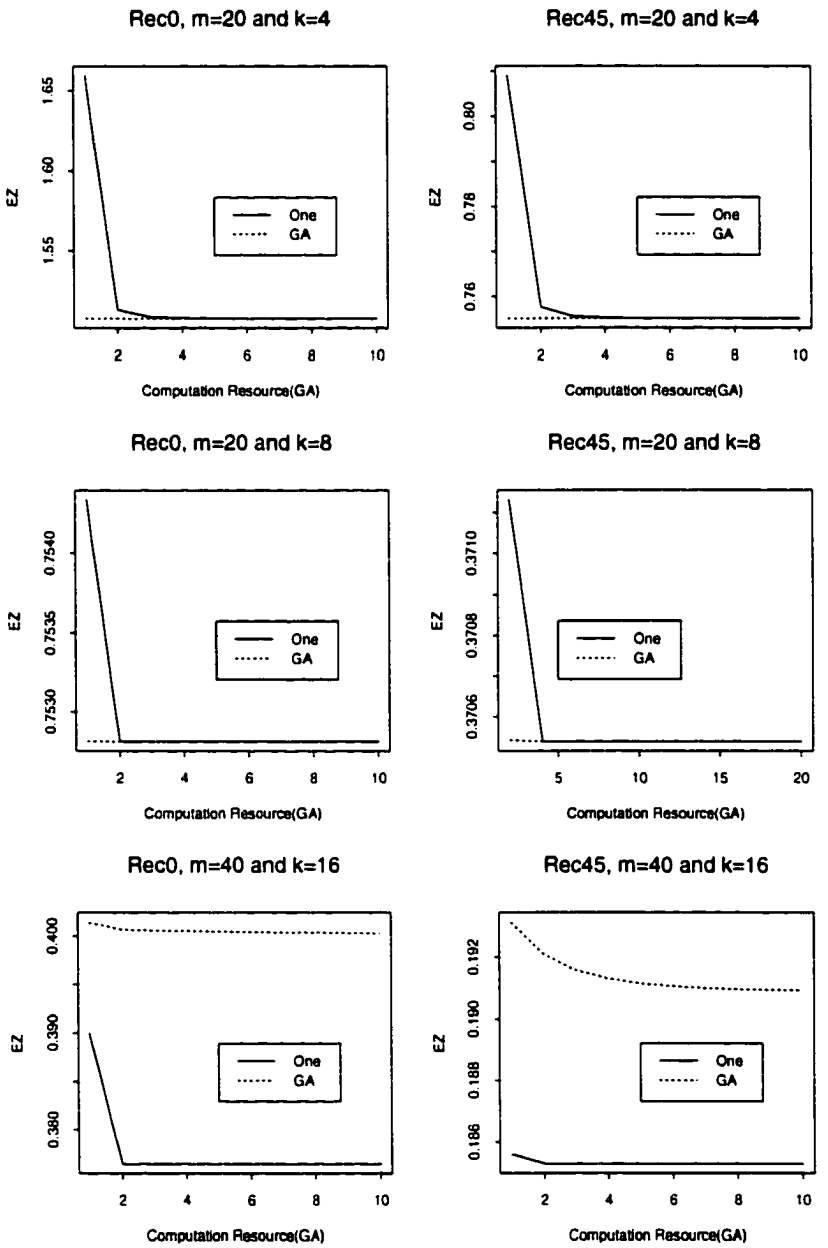

Figure E.2. $E(Z_n)$ versus n (Rectangle Shape combinations)

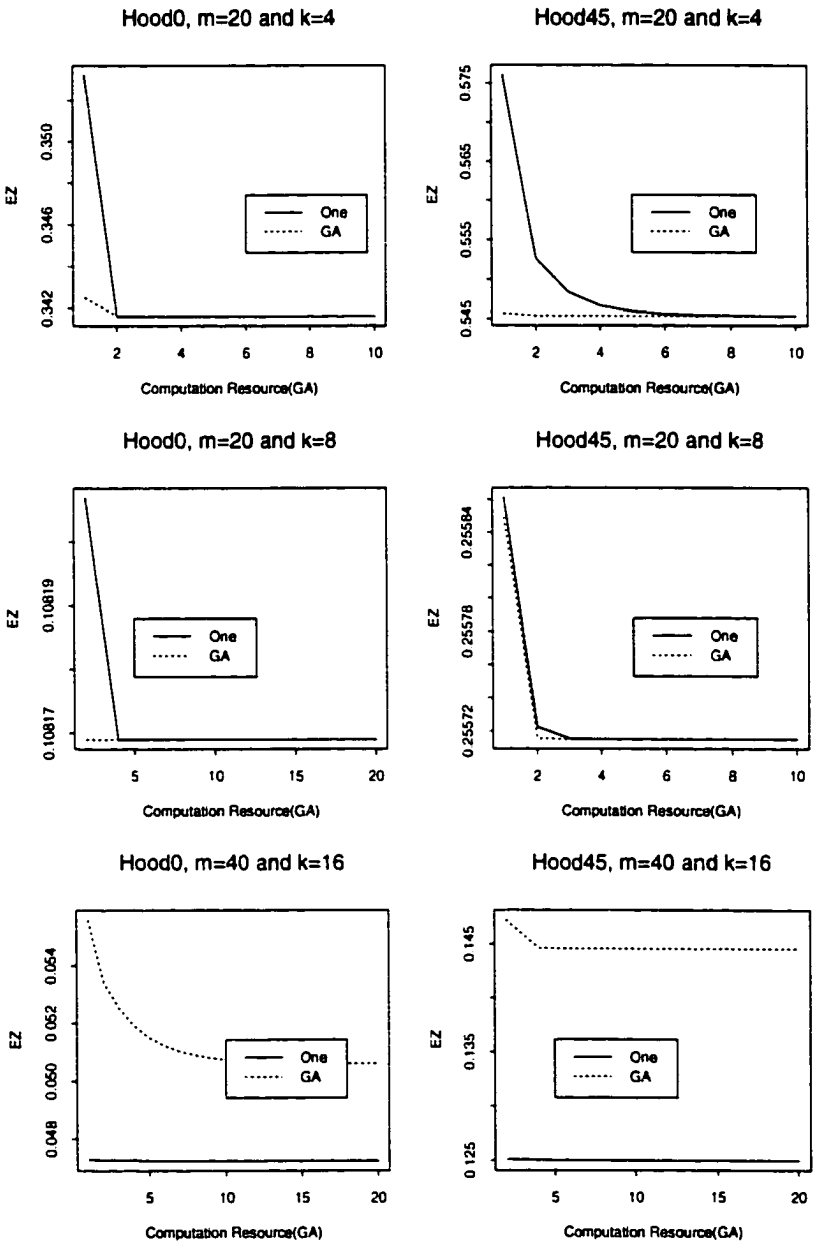

Figure E.3. $E(Z_n)$ versus n (Hood Shape combinations)

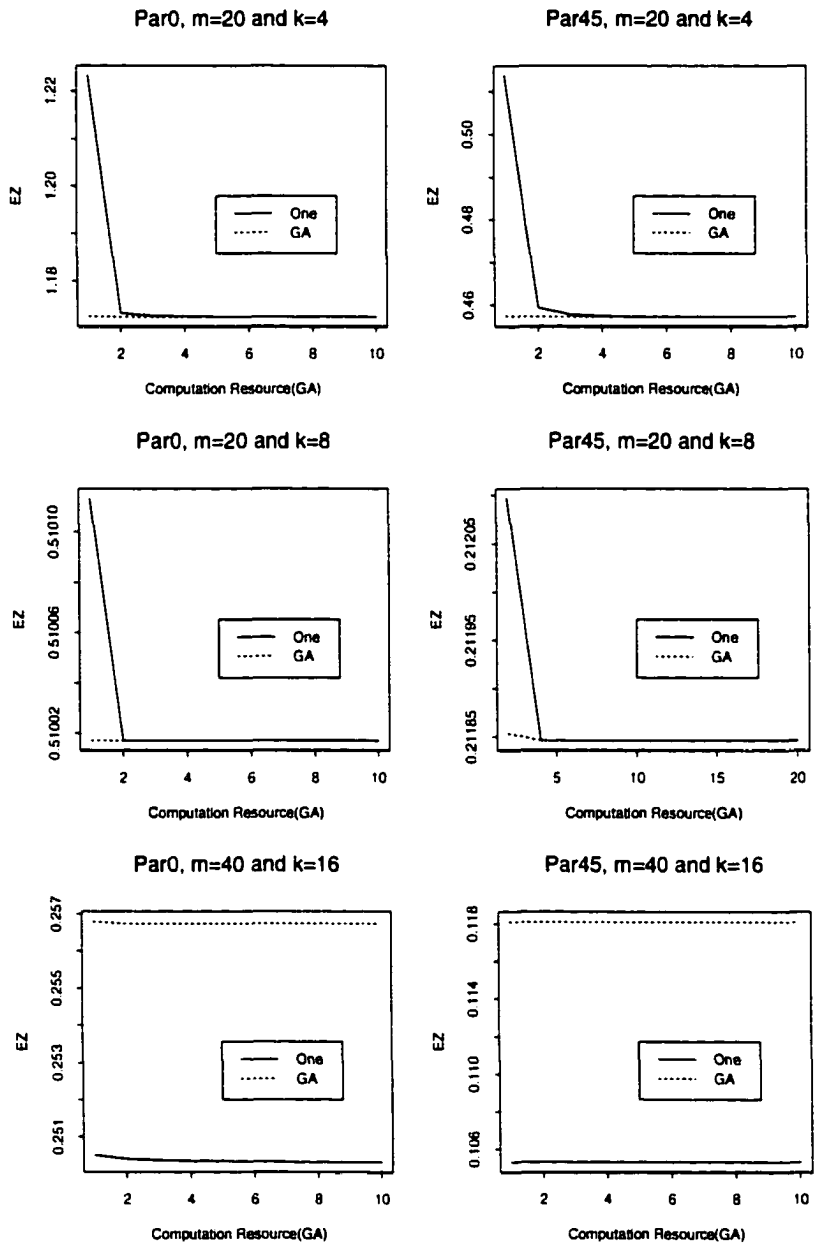

Figure E.4. $E(Z_n)$ versus n (Parabolas Shape combinations)

BIBLIOGRAPHY

Dowling, M.M., Griffin, P.M., Tsui, K.- L., Zhou.C. (1997), "Statistical Issues in Geometric Feature Inspection Using Coordinate Measuring Machines," *Technometrics*, Vol.39, No.1, pp.3-17.

Hulting, F.L. (1992), "Methods for the Analysis of Coordinate Measurement Data," *Computing Science and Statistics*, Vol.24, pp.160-169.

Hulting, F. L. (1995), "Comment: An Industry View of Coordinate Measurement Data Analysis," *Statistica Sinica*, Vol.5, pp.191-204.

Hulting, F.L. (1997), " Comment on Statistical Issues in Geometric Feature Inspection Using Coordinate Measuring Machines," *Technometrics*, Vol.39, No.1, pp.18-20.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs, Third Edition.* Springer-Verlag, Berlin, Heidelberg, and New York.

Seber, G. A. F., and Wild, C. J. (1989). *Nonlinear Regression.* John Wiley & Sons Inc., New York.

Wang, Y., Gupta, S., Hulting, F.L., Fussell, P.S. (1998), " Manufactured Part Modeling for Characterization of Geometric Variations of Automotive Spaceframe Extrusions," *Journal of Manufacturing Science and Engineering*, Vol.120, August 1998, pp.523-531.