Estimating Values of Complex Dependencies in COVAMOF

MASTERS THESIS

John Businge

Department of Mathematics and Computing Science, University of Groningen,
P.O. Box 800, 9700 AV, The Netherlands
johnxu21@yahoo.com

August 07, 2006

Supervisors: Marco Sinnema and Sybren Deelstra

I dedicate this work to my beloved parents, girlfriend, and siblings.

Abstract

Complex dependencies in COVAMOF (Configuration of Industrial Product Families Variability Modeling Framework) are dependencies that are affected by a large number of variation points. In most cases, they may not be stated formally, but have some informal character, e.g. "the combination of parameter settings will have a negative effect on the performance of the overall software system". In this thesis, we use mathematical algorithms of interpolation and extrapolation to try to formalize the determination of complex dependency values. We specifically look at complex dependency values that are affected by the variations in two and three value bindings. For dependency values that are affected by variations of two value bindings, we compare estimates obtained by interpolation/extrapolation algorithms like 3-term Level plane, 4-term Bilinear, 8-term Biquadratic, 9-term Biquadratic, and 16-term Bicubic equations. For dependency values that are affected by three value bindings, we compare estimates obtained by techniques like 7-term Trilinear, 8-term Trilinear, 27-term Triquadratic, and 64-term Tricubic.

Our findings from the data set used show us that there is no dominating algorithm in producing better estimates. Lower order algorithms like 3-term Level plane and 4-term Bilinear for 2 dimensions and 7-term Trilinear and 8-term Trilinear seems a well thought alternative since they use the fewest data points and have the smallest execution time in their computations.

Acknowledgements

I would like to acknowledge the financial support from NUFFIC and the coordinators of ICT capacity building project in Uganda and in the Netherlands for giving me the opportunity of realizing my dream.

I would also like to thank my supervisors Mr. Deelstra Sybren and Mr. Sinnema Marco for their direction, assistance, and guidance. Completing this thesis would not have been possible in isolation from them. In addition, the contribution of Prof. Wubs F.W. has been invaluable in writing this thesis.

I also would like convey warm thanks to my classmates especially Doreen Tuheirwe and John Kizito. Discussing with them in most of the course we did together plus their friendship kept me feel at home in the Netherlands.

Many thanks also go to my parents Mr. and Mrs. Musinguzi for the support they gave me since my childhood. Finally, Words alone cannot express the thanks I owe Catherine Asasira for her patience and words of encouragement for the two years we have not been together.

John Businge

Table of Contents

1 Introduction 1

1.1 Outline 1

2 Background 2

2.1 Variation Points, Dependencies, and Tool Support 3

2.1.1 Variation Points 4

2.1.2 Dependencies 4

2.1.3 Tool-Support 6

3 Problem Statement 10

3.1.1 Scope 12

4 Methods Used 13

4.1 Interpolation and Extrapolation Techniques 13

4.2 Interpolation/Extrapolation in 2D 13

4.2.1 Linear Plane Interpolation/Extrapolation 14

4.2.2 Bilinear Interpolation/Extrapolation 14

4.2.3 Biquadratic Interpolation/Extrapolation 15

4.2.4 Bicubic Interpolation/Extrapolation 16

4.3 Interpolation/Extrapolation in 3D 17

4.3.1 Trilinear Interpolation/Extrapolation 18

4.3.2 Triquadratic Interpolation/Extrapolation 19

4.3.3 Tricubic Interpolation/Extrapolation 19

4.4 Summary 19

5 Mapping of the Different Techniques to COVAMOF 21

6 Evaluation and Results of the Algorithms 24

6.1 Description of the parameters used 24

6.2 Results 25

6.2.1 2D Results 25

6.2.2 3D Results 26

6.3 Evaluation 26

6.3.1 Comparison of closer and distant data points 28
7 Summary and Conclusion 29
8 References 31
9 Appendix A 33
9.1 2D interpolation results 33
9.2 2D Extrapolation results 34
9.3 3D interpolation results 35
9.4 3D extrapolation results 36
9.5 2D interpolation RMSE 37
9.6 2D extrapolation RMSE 38
9.7 3D interpolation RMSE 39
9.8 3D extrapolation RMSE 40
10 Appendix B 43
10.1 Graphs for different underlying function in 2D 43
10.2 Graphs for the comparison of the different algorithms in 2D 44
10.3 Graphs for the comparison of the different 3D algorithms 48
11 Appendix C 53
11.1 Function for parsing xml file 53
11.2 Function for 2D sorting 54
11.3 Function for calculating the coefficients of an equation 55
11.4 Function for estimating a point in 2D space 56

List of Figures

Fig. 1. Graphical notations of variation points, variants, and dependencies in the CVV .16

Fig. 2. Graphical representation of N variation points with corresponding variants.....20

Fig. 3. Graphical representation of a dependency with N variation points.....20

Fig. 4. Level plane estimation.....24

Fig. 5. Unit square grid layout of Bilinear interpolation.....25

Fig.6. Grids for eight-term and nine-term biquadratic interpolation.....25

Fig. 7. A grid for 16-term bicubic interpolation.....26

Fig. 8. Element of interpolation in three dimension.....28

Fig. 9. Notation of reference data in COVAMOF.....31

1 Introduction

A software product family (SPF) is a set of software-intensive systems that share a common, managed set of artifacts satisfying the specific needs of a particular market segment, developed from a common set of core assets in a prescribed way [7]. Configuration of Industrial Product Families Variability Modeling Framework (COVAMOF) is an approach that uniformly models variability in all abstraction layers (from features down to code) of a software product family [15]. COVAMOF captures variability in terms of variation points and dependencies. Dependencies are conditions whose values are based on the selection of a particular set of variants or values from one or more variation points. These dependencies are separated into two different categories, i.e. simple and complex dependencies.

During product derivation from the product family, COVAMOF stores both variant or value bindings plus their corresponding dependency result in an inference engine. When a new software product is to be derived, experts are sometimes able to predict the dependency result of different variant or value bindings. Currently, prediction is only based on earlier choices, logical expressions, and simple mathematics. The purpose of this thesis is to extend the power of the inference engine with more advanced mathematics like interpolation and extrapolation.

1.1 Outline

The next chapter presents the background of this thesis. Chapter 3 explains problem statement. Chapter 4 discuses the research methods applied in this thesis. Chapter 5 explains how the different interpolation/extrapolation techniques maps onto COVAMOF. Chapter 6 presents the evaluation and results of the designed algorithms on the COVAMOF data. Chapter 7 concludes the thesis.

2 Background

Software reuse has been a key component in the software engineering community for the last few decades [1]. Reuse facilitates increased productivity, quality, and reliability, and the decrease of costs and time-to-market. The idea to develop product families originated in the 1970s and has evolved into practical approaches that promote reuse of core artifacts across related software products [2]. An example of such an approach is the software product family [3], which is an approach to reuse that involves creating a collection of similar software systems from a reusable set of software artifacts. A software product family typically consists of three abstraction layers, i.e. features layer, architecture layer, and component implementation.

Deelstra et al. in [19] states that the basic philosophy of software product families is intra-organizational reuse through the explicitly planned exploitation of commonalities between related products. The maturity levels of software product families are a useful framework for companies to deploy in order to increase their product family maturity, [8], [5], [7], and [6]. The different levels according to the scope of reuse include:

- *Standardized infrastructure.* To maximize reuse, individual software products should be built on a standardized infrastructure that consists of operating system, database manager, graphical user interface, development tools, etc.
- *Platform:* The next level in reuse after setting up a standardized infrastructure is a platform that uses this infrastructure. The platform should also consist of artifacts that capture the domain specific functionality common to all products.
- *Software product line:* After maintaining a platform that uses the standardized infrastructure, the next level is to set up a software product line where all products share a common, managed set of artifacts in a given domain. Variation points are added so that individual products can be derived from software product line.
- *Configurable product family:* The last level in reuse is when no effort is necessary when deriving a software product from a software product family

in terms of design and programming. Individual software products can be derived from a configurable software product family even by non-experts

Software product family engineering is based on two key principles: (1) the differentiation between two development processes: domain engineering and application engineering, and (2) the explicit definition and management of the product line variability. During domain engineering, the differences in a software product family are defined and the commonalities in the domain artifacts (e.g. components and classes) are developed. During application engineering, the variability in software product families is exploited by binding the variants that are required for the customer-specific application [4].

Variability is a central concept in software product family engineering. Software architects try to prepare the architecture of a software product family for product diversification, i.e. preparing the family architecture for different product contexts, by implementing variability: variability can be defined as the ability of a software or artifact to be extended, changed, customized or configured for use in a particular context [4]. According to Cox et al in [18], Examples of variability taking an example of a banking system for all banks and specifically the ATM subsystem could include:

- Varying a greeting for display on an ATM, e.g. displaying a greeting with different bank names.
- Varying the a language of choice for display to the client, e.g. English or Spanish
- Varying an action if an ATM card has expired, e.g. confiscating the card or ejecting the card.

2.1 Variation Points, Dependencies, and Tool Support

In the first subsection, an explicit representation of variability in software product families plus how variation points and associated dependencies affect its manageability is stated. In this subsection, a detailed definition of variation points and dependencies are

explained. The subsection further states the advantage of tool-support for the consistent documentation and management of variability.

2.1.1 Variation Points

Variation points are places in a design or implementation that identify locations at which variation occurs [10]. The different options in variation points that can be bound are represented by a value, or by a set of variants that are associated to the variation point. To facilitate the communication of variation points and their associated variants to the user, both the variation points and the variations must be explicitly represented in all the three levels of abstraction (i.e. features, architecture, and component implementation). This eases systematic documentation and traceability of variability, development for and with reuse [11]. Sinnema et al [11], categorizes five types of variation points as:

- An *optional* variation point is the choice between zero or one from the one or more associated variation points.
- An *alternative* variation point is the choice between one of the one or more associated variants.
- An *optional variant* variation point is the selection (zero or more) from the one or more associated variant.
- A *variant* variation point is the selection (one or more) from the one or more associated variants.
- A *value* variation point is a value that can be chose in a predefined range.

These five types of variation points can further be categorized into two main types, that is, variant and value variation points.

2.1.2 Dependencies

Sinnema et al. in [7], defines dependencies as conditions whose values are based on the selection of a particular set of variants for one or more variation points (e.g. for consistency, compatibility, or quality attributes). They represent system functionality and system properties such as safety, security, reliability and usability [11] and specify how the binding of variation points influences the value of the system property. As the number of products in a product family tends to grow, variability infrastructure becomes more

fine-grained and therefore more complex. The number of variation points for industrial product families may range in the thousands, which is already a challenging number to manage, but the number of dependencies resulting from the binding of these variation points typically has an exponential relationship [13]. The need to consider and manage dependencies between different variation points right from the requirements level should be considered for properly managing variability.

Sinnema et al. in [9] and [15] categorize dependencies as simple dependencies and complex dependencies: Simple dependencies specify the restriction on the binding of one or two variation points. They may include, but not limited to:

- *Excludes-dependency:* describes the binding of one variant excludes the selection of another variant (excluded variant), expressed a long the line of "the binding of variant A1 to variation point A excludes the binding of variant B1 of variation point B" – means that only one of those variants can be selected e.g. when you are buying a roof-less car, you cannot choose a sunroof.
- *Requires-dependency*: describes the binding of one variant implies the need of another variant (required variant), expressed a long the line of "the binding of variant A1 to variation point A requires the binding of variant B1 of variation point B". For example if one wants to lock up his car via remote control this variant requires a centralized door locking.

Complex dependencies are typically affected by a large number of variation points. In most cases, they may not be stated formally, but have some informal character, e.g. "the combination of parameter settings will have a negative effect on the performance of the overall software system". This is mainly due to the restrictions or constraints imposed on the combination of variants that are selected at each of the variation points during the derivation of a software product.

Some examples of complex dependencies could include:

- *Maximum response time:* For a software product to attain certain degree of *Performance*, the response time of this software product must be restricted

to a certain maximum value. The response time is a result of a combination of different variants at each of variation points which can not be easily determined by simple in- or exclusion of variants and therefore referred to as complex dependency *maximum response time*.

- *Maximum failure rate:* Similarly, for a software product to attain a certain degree of *safety*, the error rate must be restricted to a certain maximum value. This failure rate is also a result of a combination of different variants at each of the variation points that can not determined by simple in- or exclusion of variants and therefore referred to as complex dependency *maximum failure rate*.

2.1.3 Tool-Support

Because of the very large numbers of variation points to choose from and dependencies that exist on the choices, the configuration of software products from software product families can be a very complex task. Clements et al. [7] states that configuration is both error-prone and time-consuming: This is mainly because the consistent documentation and management of product family variability still is a challenge in software product family engineering. To deal with these problems, the authors of [9] and [11] state that an adequate tool support is the solution and therefore they developed a modeling framework COVAMOF and a supporting tool-suite for Microsoft Visual studio .NET [16] (COVAMOF-VS). This framework consists of modeling facilities that model the variation points and dependencies uniformly over different abstraction levels (e.g. features, architecture and implementation), as first-class citizens.

In trying to model the variation points and dependencies, COVAMOF enables providing different views on the variability provided by product family artifacts, i.e. the COVAMOF variability view (CVV), which constitutes of the variation point view, and the dependency view.

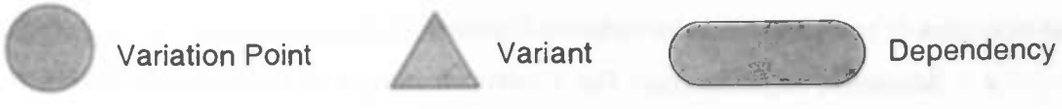


Fig. 1. Graphical notations of variation points, variants, and dependencies in the CVV

2.1.3.1 Variation point View

The variation point view shows the variation points in the product family. The main purpose of this view is to show an engineer which choices are available at different abstraction layers, how they realize each other across layers, and how choices depend upon each other. The view contains the following entities: Variation Point, Variant, Realization and Dependency:

- *Variation Point:* this represents a point in which choices or options for configurations of individual software products are offered in a product family.
- *Variant:* These are the different options in a variation point that are available choice during configuration of individual products from a software product family
- *Dependency:* COVAMOF variability model captures both simple and complex dependencies (see also subsection 2.1.2). The dependency maps the binding of variation points to a value of a property such as maximum processing time or performance.

2.1.3.2 Dependency View

The dependency view in CVV contains both dependencies and dependency interaction. The main purpose of this view is to show how dependencies interact with each other and, how an engineer can cope with these interactions. The dependences in this view are associated by one or more variation points and restrict the selection of the variants associated to these variation points. The view captures both simple and complex dependencies (see also subsection 2.1.2). The dependency interactions specify how two or more dependencies mutually interact. The interactions provide a description of the origin of the interaction and specify how to cope with the interaction during product derivation.

In [15], they further point out that the two ways of capturing knowledge about how the configuration of associated variation points maps to a value in the target domain, i.e. by *Association* entities and *Reference Data* entities.

- *Association:* Here associations refer to the variation points whose configuration affects the value of the dependency. Their framework distinguishes between three types of associations.
    - *Abstract:* with this association, experts only know that the variation point has an effect on the value of the dependency but do not know to what extent the impact will be.
    - *Directional:* With this type of association, experts do not necessarily know the effect of the variation point on the value of the dependency but at least they know the direction of the change, i.e. whether the variations in the value bindings will increase or decrease the value of the dependency.
    - *Logical:* With this association, experts completely know the effect in which changes in the variation point will have on the dependency value.
- ***Reference Data:*** The second way in of capturing knowledge about how the configuration of associated variation points maps on to the value of the dependency is by modeling reference data. Reference data entities contain measurements of the value of a dependency for specific configurations of the associated variation points. These entities contain measurements of the associated variation points. These measurements originate from tests of products that have been derived from the product family.

During product derivation, engineers use the three types of knowledge plus the reference data in two ways: On the one hand, they can be used for finding the starting points for the derivation process - when a reference data element is selected whose specific system property value is closest to the required value for the product being derived. The bindings of this reference data element can be used as a starting point in the initial phase, where a first configuration of the product is derived and when the configuration is not finished, the product enters the iteration phase where the configuration is changed until the product is ready. On the other hand, they can be used during derivation process to estimate the value of the system property bases on the binding of variation points.

According to [13], the engineers who develop the architecture of the product family generally have a concept for what to do when a particular product is needed and also understands its implications. If these engineers are involved in the derivation of required products, the results can always be probably sufficient. However, the adapters of the architecture are typically not always the creators and this may result in laborious, time-consuming, and error-prone that requires a number of iterations before a product is finished and hence less optimal solutions. Therefore, the purpose of this thesis is to develop new methods for this problem.

3 Problem Statement

Because complex dependencies are often indirectly implied, not documented, and sometimes unintended, they are most of the time very difficult to model. Like the two examples of complex dependencies that were presented in section 2.1.2, achieving their optimum values can be time consuming and requires performing a number of iterations.

In subsection 2.1.3.2, we stated how COVAMOF tries to overcome this problem by use of the two ways in which it captures knowledge, that is, *Associations* and *Reference Data*. Using this knowledge, with a reselection of variants at the different variation points, experts can in some cases be able to predict the influence of this reselection on the dependency and therefore provide reasonable estimates. In addition, this can be used as starting points of the derivations, which are thereafter refined in the iteration phase.

In other cases, due to the complexity of the problem domain (e.g. configurations which require variants from two or more variation points), experts may have no ability to predict the influence of the reselection of these variants on the dependency and therefore they are not in position to give reasonable estimates. The validation of such dependencies can only be determined by testing the whole software product. Sinnema et al. in [11] and [15], refers these types of dependencies as *Dynamically analyzable Dependencies.*

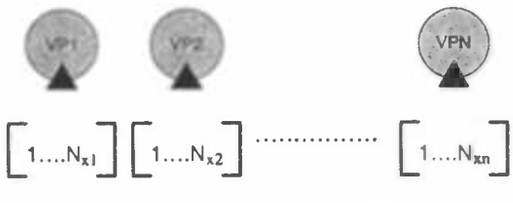


Fig. 2. Graphical representation of N variation points with the corresponding variants

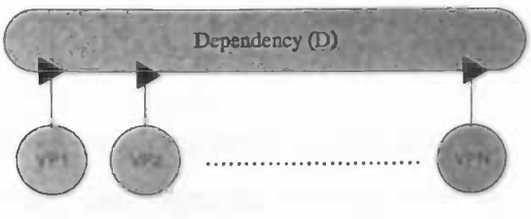


Fig. 3. Graphical representation of a dependency with N variation points

The value of a complex dependency, D, depends on selected values from each of the N variation points (see also Figure 2 and 3 above), that is,

- VP1 having an array of values v1a [1…N_{x1}]
- VP2 having an array of values v2a [1…N_{x2}]

- .....
- VPN having an array of values vNa [1...N_{xn}]

A number of configurations for different software products are made and values of dependency plus its corresponding values from the different variation points are stored in the inference engine. As stated earlier, the data in the inference engine would be the reference data.

If new software products require some configuration of values from the different variation point already existing in the inference engine, then the value of the dependency can be predictable. The result is unpredictable if the software product requires different configuration of the values.

If the variations are not so many, like one or two values from the different variation points changed, engineers might be able to give a first estimate that can be refined in the iteration phase. Otherwise, the whole software product would have to be tested to determine the value of the dependency.

The research problem of this thesis is to develop algorithms that will automate the determination of estimate values of complex dependencies by the help of interpolation and extrapolation taking the reference data entities as inputs.

With Interpolation, the value of the dependency being estimated should lie between values already known or tabulated, that is,
If we want to estimate of the value of function D at some untabulated point $(v_1, v_2, ..., v_N)$, then the values $v_1, v_2, ..., v_N$ should fall in the range of the tabulated values (reference data) stored in the arrays v1a, v2a, ... ,vNa respectively [14], that is,

$$v1a[i_1] \le v_1 \le v1a[i_1 + 1]$$
$$v2a[i_2] \le v_2 \le v2a[i_2 + 1]$$

.....

$$vNa[i_N] \leq v_N \leq vNa[i_N + 1]$$

This implies that to be able to interpolate the dependency D with N variation points, the minimum number of reference data points stored in the inference engine should be equal 2^N, i.e.

- For 2 variation points, we should have 4 reference data points
- For 3 variation points, we should have 8 reference data points, etc.

The relation of the input quantities (reference data), to an underlying function

$D\ (v_1, v_2, ..., v_N)$ is

$$D\ [i_1]\ [i_2]\ ....\ [i_N] = D\ (v1a\ [i_1], v2a\ [i_2], ..., vNa\ [i_N])$$

Cited in Press et al. [14].

When the reference data points are less than 2^N for the N variation points, then this would be an extrapolation problem (estimation of the value of the dependency outside the range of known or tabulated values). The error in the value being extrapolated may sometimes be very unpredictable since extrapolation can diverge exponentially from the underlying function (i.e. it is possible that the function can easily run to infinity outside the range). The validity of value being extrapolated will also heavily depend on the how close it is from points necessary for interpolation (the further the points, the less accurate the estimate will be) [14][24].

3.1.1 Scope

In subsection 2.1.1 two types of variants were defined. Variants, which include: *optional variation, alternative, optional variant and variant,* may represent anything from an object or class, to file, or code-block. Value variants are used to represent parameters (see also 2.2.3.1).

This thesis will only look at values and will not look at the variants because it is not easy, if not impossible, to quantify none parameters like a file or an object mathematically using interpolation.

4 Methods Used

In the previous section, we stated the two approaches — interpolation and extrapolation — that can be used to estimate values of complex dependencies. This section, presents the different interpolation and extrapolation techniques that can be used for this purpose.

4.1 Interpolation and Extrapolation Techniques

Interpolation can be defined by the following example according to [14]:
"We sometimes know the value of a function $f(x)$ at a set of points $x_1, x_2, ..., x_N$ (say, with $x_1 < ... < x_N$), but we don't have an analytic expression for $f(x)$ that lets us calculate its value at an arbitrary point. For example, the $f(x_i)'s$ might result from some physical measurement or from long numerical calculation that cannot be cast into a simple functional form. Often the $x_i's$ are equally spaced, but not necessarily. The task now is to estimate $f(x)$ for arbitrary x by, in some sense, drawing a smooth curve through (and perhaps beyond) the x_i. If the desired x is in between the largest and smallest of the $x_i's$, the problem is called *interpolation*; if x is outside that range, it is called e*xtrapolation*."

As stated earlier in section 3.0, extrapolation is the process of constructing new data points outside the range of known data points. The validity of the value being extrapolated largely depends on the original data. If the data is smooth, the results of extrapolation can be reliable to a given extent otherwise the results are often less meaningful, and are subject to greater uncertainty.

4.2 Interpolation/Extrapolation in 2D

Schumaker in [26] states that, "Given the points (x_i, y_i, z_i), for $i = 1, 2, ..., n$, over some domain, a function $z = f(x, y)$ is desired which reproduces the given points, and produces a reasonable estimate of the surface (z) at all other points (x, y) in the given domain" . The equation below can be used to estimate any point with known coordinates.

$$h_i = \sum_{k=0}^{m} \sum_{l=0}^{n} a_{kl} x^k y^l \qquad (3.1)$$

where h_i is the height at a point i, x and y are rectangular coordinates of i, and a_{kl} are the coefficient of the polynomial. The values of a can be determined by a set of simultaneous equations that set up, one for each point of the known rectangular coordinates.

Various polynomial interpolants of different degrees can be considered from equation 3.1 above for estimation of any unknown point (x, y). In this thesis, we will compare estimates of linear plane (1-degree), bilinear (1-degree), biquadratic (2-degrees), and bicubic (3-degrees) polynomials.

4.2.1 Linear Plane Interpolation/Extrapolation

For the linear plane, a three-term polynomial formed by fitting a triangle to the three closest known grid vertices.

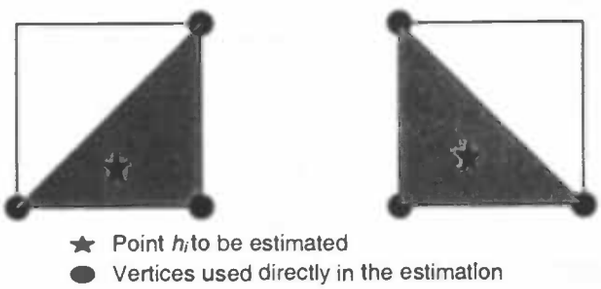


Fig. 4 Level plane estimation.

$$h_i = a_{00} + a_{10} x + a_{01} y \qquad \ldots (3.2)$$

With three known closest grid vertices, a set of three equations can be formed which can be solved simultaneously to determine the coefficient of equation 3.2. The point to be estimated with known coordinates (x, y) can be evaluated by just substituting its coordinated in equation 3.2.

4.2.2 Bilinear Interpolation/Extrapolation

Bilinear interpolation/extrapolation is the type of interpolation that can be applied on data points in two dimensions. It is an extension of the one dimension linear interpolation on a straight line onto a grid/plane. The height of the point being interpolated can be

calculated by comparing its coordinates on the plane with those of the four closest data points [14], [20]. In [20], bilinear interpolation is exemplified by the Fig. 5.

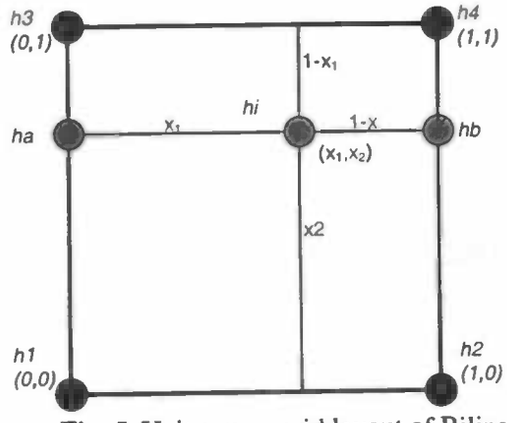


Fig. 5. Unit square grid layout of Bilinear interpolation

The bilinear function is akin to fitting a hyperbolic paraboloid to the four vertices of the grid cell. The function is a four-term polynomial derived from equation (3.1) by substituting both m and n with 1.

$$h_i = a_{00} + a_{10}x + a_{01}y + a_{11}xy \qquad \ldots (3.3)$$

or in more compact form

$$h_i = \sum_{j=0}^{1}\sum_{k=0}^{1} a_{jk} x_1^j x_2^k \qquad \ldots (3.4)$$

With four known vertices, a set of four equations can be formed that can be solved simultaneously to determine the coefficients of equation 3.3.

4.2.3 Biquadratic Interpolation/Extrapolation

Bilinear interpolation can be extended to biquadratic by considering the mid-points of the grid square in Fig. 5.

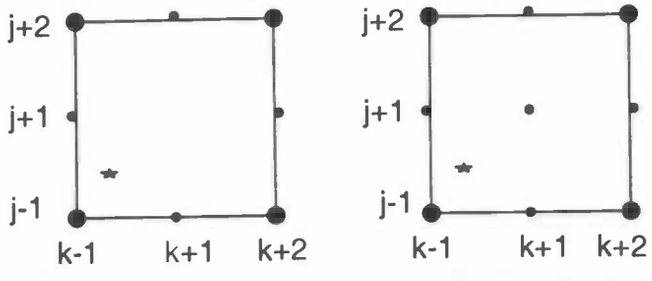


Fig. 6 grids for eight-term and nine-term biquadratic interpolation

The biquadratic polynomial can be derived from equation 3.1 by considering up to the quadratic terms.

$$h_i = a_{00} + a_{10}x + a_{10}y + a_{20}x^2 + a_{11}xy + a_{02}y^2 + a_{21}x^2y + a_{22}x^2y^2 \quad \ldots (3.5)$$

In more compact form,

$$h_i = \sum_{j=0}^{2}\sum_{k=0}^{2} a_{jk}x^j y^k \quad \ldots (3.6)$$

the ninth term x^2y^2 can also be omitted to form an eight-term biquadratic polynomial.

$$h_i = a_{00} + a_{10}x + a_{10}y + a_{20}x^2 + a_{11}xy + a_{02}y^2 + a_{21}x^2y + a_{12}xy^2 \quad \ldots (3.7)$$

4.2.4 Bicubic Interpolation/Extrapolation

Bicubic interpolation is one of the higher orders for smoothness most widely used interpolation technique when the 2D data points are not linearly distributed. The point being interpolated is a combination of 16 closest data points.

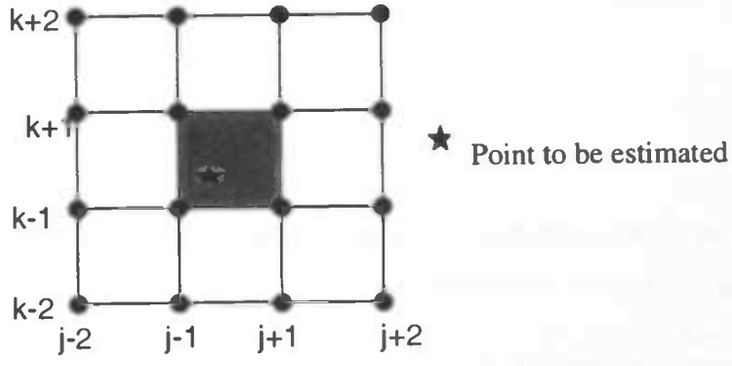


Fig. 7 a grid for 16-term bicubic interpolation

Bicubic interpolation searches for a function of the form

$$f(x_1, x_2) = \sum_{j=0}^{3}\sum_{k=0}^{3} c_{jk}x_1^j x_2^k \quad \ldots (3.7)$$

Using bicubic interpolation, requires that at each grid point the function $y(x_1, x_2)$, the gradients $\partial y/\partial x_1 \equiv y_{,1}$, $\partial y/\partial x_2 \equiv y_{,2}$ and the cross derivative $\partial^2 y/\partial x_1 \partial x_2 \equiv y_{,12}$ to be continuous (i.e. they should be defined at each grid points) [22].

In **Fig. 7** above if we imagine that the data points are stored in a matrix of functional values ya[j] [k], where j varies from 1 to m, and k varies from 1 to n and also that the values of x_1 and x_2 are also stored in arrays x1a of length m and x2a of length n respectively. Then

h_1 = ya[j] [k]

h_2 = ya[j+1] [k]

h_3 = ya[j+1] [k+1]

h_4 = ya[j] [k+1]

The relevant code for determining the gradients and cross derivative at the data point h_1 using numerical differencing would be something like:

```
y1a[j] [k] = (ya[j+1] [k]-ya[j-1] [k] )/(x1a[j+1]-x1a[j-1]);
y2a[j] [k] = (ya[j][k+1]-ya[j][k-1])/(x2a[k+1]-x2a[k-1]);
y12a[j] [k] = (ya[j+1][k+1]-ya[j+1][k-1]-ya[j-1][k+1]+ya[j-1][k-1])/
        ((x1a[j+1]-x1a[j-1])*(x2a[k+1]+x2a[k-1]));
```

Where y1a, y2a, and y12a are the gradients and the cross derivative respectively at data point h_1. For the gradients and the cross derivative to be continuous/defined, the values of arrays x1a and x2a must be varying monotonically (increasing or decreasing). In other words, the denominators should never be zero.

4.3 Interpolation/Extrapolation in 3D

The concept of 2D interpolation/extrapolation can be extended to 3D by considering data points located at the vertices of a box. The equation used for estimation of any point with known vertices is given as:

$$p_i = \sum_{j=0}^{n_1}\sum_{k=0}^{n_2}\sum_{l=0}^{n_3} a_{jkl} x^j y^k z^l \qquad \ldots (3.8)$$

where p_i is the volume of at point i, x, y and z are box coordinates and a_{jkl} are the coefficients of the polynomial.

Like for 2D above, various polynomial interpolants of different degrees can be considered from equation (3.8) above for estimation of any unknown point (x, y, z). In this thesis, we will compare estimates from trilinear (1-degree), triquadratic (2-degrees), and tricubic (3-degrees) polynomials.

4.3.1 Trilinear Interpolation/Extrapolation

Trilinear interpolation is a type of interpolation technique that is applied to data points in three dimensions. It is also an extension of the two dimension bilinear interpolation on a grid/plane to a box (3D)/hyper plane. The data points in trilinear interpolation are located at the vertices of the box. The point being estimated is a combination of eight closest data points. Perhaps its most common application is interpolating within cells of a volumetric dataset [20], [21].

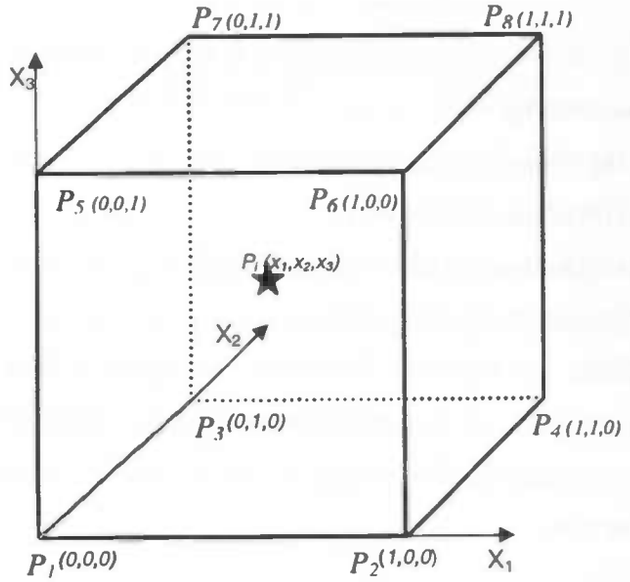


Fig. 8 Element of interpolation in three dimensions

The value at position (x_1, x_2, x_3) within the cube will be denoted P and is given by:

$$p_i = a_{000} + a_{100}x_1 + a_{010}x_2 + a_{001}x_3 + a_{110}x_1x_2 + a_{101}x_1x_3 + a_{011}x_2x_3 + a_{111}x_1x_2x_3 \quad \ldots (3.9)$$

or in more compact form

$$p_i = \sum_{l=0}^{1}\sum_{m=0}^{1}\sum_{n=0}^{1} a_{lmn}x_1^l x_2^m x_3^n \quad \ldots (3.10)$$

With eight data points known, equation 3.9 yields 8 equations which can then be solved simultaneously to determine the coefficients.

We will also consider estimates from a seven-term polynomial by eliminating the $x_1x_2x_3$ -term, that is,

$$p_i = a_{000} + a_{100}x_1 + a_{010}x_2 + a_{001}x_3 + a_{110}x_1x_2 + a_{101}x_1x_3 + a_{011}x_2x_3 \quad \ldots (3.11)$$

4.3.2 Triquadratic Interpolation/Extrapolation

Triquadratic interpolation is an extension of trilinear interpolation by introducing the mid-points at each face of the box in Fig. 8 above. The polynomial that searches for triquadratic interpolation is of the form:

$$p_i = \sum_{j=0}^{2}\sum_{k=0}^{2}\sum_{l=0}^{2} a_{jkl} x^j y^k z^l \qquad \ldots (3.12)$$

Equation (3.12) is a 27-term triquadratic polynomial.

4.3.3 Tricubic Interpolation/Extrapolation

Another widely used interpolation technique for higher order of smoothness in three-dimension is tricubic interpolation. The point being interpolated is a combination of 64 closest data points. This interpolation procedure also maintains continuity of the function and its first derivatives across cell boundaries (e.g. the vertices of the box in **Fig.** 8) [23]. Just like the bicubic interpolation, tricubic interpolation also requires that at each grid point the function $y(x_1, x_2, x_3)$, the gradients $\partial y/\partial x_1 \equiv y_{,1}$, $\partial y/\partial x_2 \equiv y_{,2}, \partial y/\partial x_3 \equiv y_{,3}$ and the cross derivative $\partial^3 y/\partial x_1 \partial x_2 \partial x_3 \equiv y_{,123}$ be defined.

The tricubic interpolation searches for a function of the form

$$f(x_1, x_2, x_3) = \sum_{i,j,k=0}^{3} a_{ijk} x_1^i x_2^j x_3^k \qquad \ldots (3.13)$$

Equation (3.13) is a 64-term tricubic polynomial. With 64 data points known, equation (3.13) yields 64 equations that can be solved simultaneously to determine the coefficients.

4.4 Summary

Table 1 presents the summary of the different interpolation/extrapolation discussed in section 4.2 and 4.3

Interpolation technique	**Rectangular data points**
2D Estimation	
3-term Linear Plane	3
4-term Bilinear	4
8-term Biquadratic	8

9-term Biquadratic	9
16-term Bicubic	16
Polynomial of degree *N*	$(N+1)^2$
3D Estimation	
7-term Trilinear	7
8-term Trilinear	8
27-term Triquadratic	27
64-term tricubic	64
Polynomial of degree *N*	$(N+1)^3$
N Dimensional Estimation	
N-Linear	2^N
N-Quadratic	3^N
N-Cubic	4^N
Polynomial of degree M	$(M+1)^N$

Table 1. Summary of interpolation/extrapolation

Hypothesis: Because Higher-order interpolation/extrapolation techniques use more data points enclosing the point being estimated, they will always be more accurate than lower-order techniques.

5 Mapping of the Different Techniques to COVAMOF

In Chapter 4, we discussed the different interpolation/extrapolation algorithms that can be used to estimate a point that lies in two or three dimension space. In this chapter, we discuss how the discussed algorithms can be used to estimate complex dependencies in COVAMOF with a given sample of reference data.

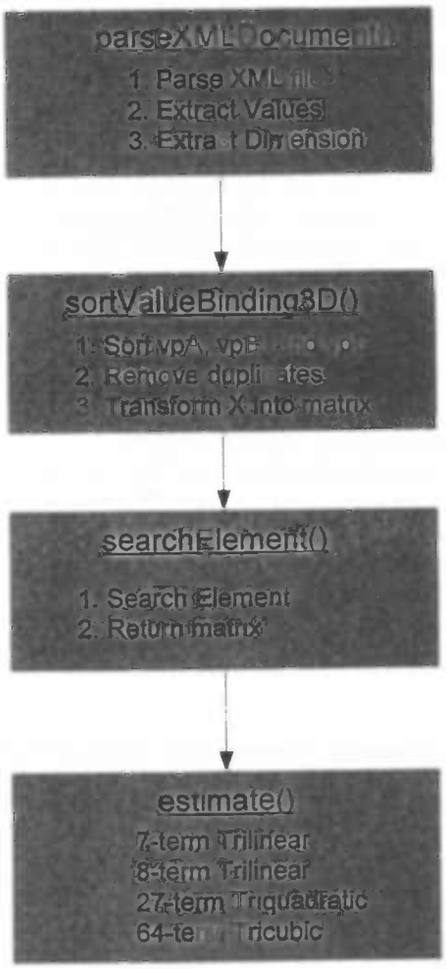


Fig.10. Summary of the steps in mapping

Reference data in COVAMOF is store in XML format. The Fig. 9 below illustrates the notation of how reference data is stored.

```
<dependency name="X">
      <referencedata result="52">
            <valuebinding vp="A" value="5"/>
            <valuebinding vp="B" value="8"/>
            <valuebinding vp="C" value="3"/>
      </referencedata>
      <referencedata result="14">
            <valuebinding vp="A" value="5"/>
            <valuebinding vp="B" value="8"/>
            <valuebinding vp="C" value="2"/>
      </referencedata>
      <referencedata result="31">
            <valuebinding vp="A" value="10"/>
            <valuebinding vp="B" value="4"/>
            <valuebinding vp="C" value="3"/>
      </referencedata>
      <referencedata result="40">
            <valuebinding vp="A" value="10"/>
            <valuebinding vp="B" value="4"/>
            <valuebinding vp="C" value="2"/>
      </referencedata>
      <referencedata result="46">
            <valuebinding vp="A" value="10"/>
            <valuebinding vp="B" value="8"/>
            <valuebinding vp="C" value="3"/>
      </referencedata>
      <referencedata result="25">
            <valuebinding vp="A" value="5"/>
            <valuebinding vp="B" value="4"/>
            <valuebinding vp="C" value="2"/>
      </referencedata>
</dependency>
```

Fig. 9 Notation of reference data in COVAMOF

To estimate the value of a complex dependency using interpolation and extrapolation techniques presented in section 3 above, a number of steps have to be done.

Step 1: The XML file is first parsed converting it into objects nodes of the programming language used. Thereafter, dependency values plus the corresponding value bindings from the different variation points are extracted from the XML file and they are stored in different arrays. The value bindings from the different variation points and the result of the dependency are all are stored in one dimensional array.

In the example of the notation in Fig. 9 above, we would have the following arrays:

```
int [] vpA = {5, 5, 5, 10,10, 5}//value bindings from variation
point A
int [] vpB = {8, 8, 4, 4, 8, 4} //value bindings from variation
point B
int [] vpC = {3, 2, 3, 2, 3, 2} //value bindings from variation
point C
int [] X = {52, 14, 31, 40, 46, 25}  // dependency results of the
value bindings from the three variation points A, B, and C
```

Step 2: The arrays vpA, vpB, and vpC are then sorted in ascending order and the corresponding positions of the values in array X are changed with respect to the sorted arrays. After the sorting is done, duplicates are removed in arrays vpA, vpB, and vpC. Array X is then transformed from one dimension to the dimension whose value is equal to the number of value bindings.

The results of Step 2 in a 3D matrix below:

		vpA			
		5		10	
		vpB		**VP2**	
vpC		4	8	4	8
	2	25	14	40	
	3		52		46

Table 2. 3D matrix sorted matrix

Step 3: Using the value binding points of the dependency whose result is to be estimated, the element in which this dependency lies is located by some searching algorithm like linear search or quick search.

Step 4: Estimate the unknown point using the different algorithms – 7-term Trilinear, 8-term Trilinear, 27-term Triquadratic and 64-term Tricubic.

6 Evaluation and Results of the Algorithms

In chapter 5, we showed how the presented interpolation/extrapolation algorithms in chapter 4 can be applied on COVAMOF data. This chapter discusses the outcomes of the application of the different algorithms on COVAMOF reference data. It furthermore presents the evaluation of the different algorithms by comparing the calculated root mean square error and graphs drawn.

Before we present the results and evaluation of the algorithms, we will first explain the parameters used in the experiments.

6.1 Description of the parameters used

The data used in evaluating the algorithms, was generated after performing a number of COVAMOF experiments. The data is based on Law Enforcement (e.g. speed limit enforcement) where high quality images from license plates are recorded for example by a live camera under different conditions (e.g. day, night, rain, fog, etc.). The camera does not only record license plate images but also records other images that are part of that environment. The FindComponent (FC) module, a piece of software that sits in this camera, sorts out images that look like license plate from the mixture of images. The FC module is also responsible for the initial separation between the background and foreground pixels in the license plate images, e.g. separating characters from the background.

Since so many images should be processed in a short period of time, processing speed is very important for the type of system used. All these requirements result in a number of complex dependencies that include:

1. **Average Processing Time (APT)**: the average processing time required to process an image
2. **Maximum Processing Time (MPT)**: the maximum processing time required to process an image
3. **LabelYesFCNo Rate (LYFCNR)**: The percentage of images that contain a valid license plate, but where FC says it does not.

4. **LabelNoFCYes Rate (LNFCYR)**: The percentage of images that do not contain a valid license plate, but where FC says it does.

The results of the above complex dependencies are affected by variations in the following value bindings:

1. **Stack-Size (SS)**: The number of alternative image scaling that are tried to find components in an image.
2. **Min Gradient (MG)**: The minimum gradient difference between neighboring pixels that determines whether there is a transition between fore- and background pixels.
3. **Add Mean Steps (AMS)**: The number of low pass filtering (blurring) steps that are used.

Dependencies APT and MPT have directional association while Dependencies LYFCNR and LNFCYR have an abstract association with respect to Stack-Size and Min Gradient (see section 2.1.3.2).

6.2 Results

We now turn to the discussion of the results for the different interpolation/extrapolation techniques presented in chapter 4 above. Dependencies in COVAMOF are affected by very large number of variation points. In most cases, most of the variation points are kept constant and only a few are varied. The result of the computed dependency is only affected by the varying variation points. In this thesis we mainly focused on dependencies that are affected by variations in two and three variation points and the rest are kept fixed at constant values.

6.2.1 2D Results

For 2D, values of the dependencies presented in section 6.1 are affected by variations in two value bindings (Stack-Size and Min Gradient) and all other value bindings are fixed to constant values. The values of the values bindings that were used to calculate the coefficients for the different equations (see section 4.2.1- 4.2.4) used in estimating an unknown point range from 5 – 10 for Stack-Size and from 2 – 10 for Min Gradient. Table 3 and 9 present sample dependency results plus their corresponding value bindings that

were obtained from COVAMOF experiments. Tables 4 – 8 present the results of interpolation that correspond to results of table 3. Tables 10 – 14 present the results of extrapolation that correspond to the results of table 9.

6.2.2 3D Results

For 3D, the values of the dependencies presented in section 6.1 are affected by variations in three value bindings (Stack-Size, Min Gradient, and Add Mean Steps) and other value bindings are fixed to constant values. The values of the values bindings that were used to calculate the coefficients for the different equations (see section 4.3.1- 4.3.3) used in estimating an unknown point range from 5 – 10 for Stack-Size, from 2 – 10 for Min Gradient, and 0 – 5 for Add Mean Steps. Tables 15 and 20 present sample dependency results plus their corresponding value bindings that were obtained from COVAMOF experiments. Tables 16 – 19 present the results of interpolation that correspond to results of table 16. Tables 21 – 24 present the results of extrapolation that correspond to those in table 19.

6.3 Evaluation

To summarize estimations results of the algorithms presented in section 6.2, calculation of the consequent root mean square error (rms. error), see equation (3.14), for a number of observations was used and we also considered the graphs drawn.

$$rmse = \sqrt{\frac{\sum_{i=1}^{n} (Y_i - \overline{Y}_i)^2}{n}} \qquad \ldots (3.14)$$

Where Y_i is the estimated result and $\overline{Y}_i$ is the result from the experiments n is the number of observations.

The rms. error tells us to what extent a given algorithm is accurate. The smaller the rms. error, the better the accuracy.

For 2D interpolation, we can see in tables 25 and 26 that there relatively a decrease in rms. error as we move down to the higher order algorithms. When we also look at graphs 5 and 6 we see that higher order algorithms have better estimates as compared to lower order algorithms. In tables 27 and 28 there is a relative decrease in rms. Error as we move down the higher order algorithms but when look at graphs 7 and 8, we see that most of the estimates of all the algorithms are out of range. Graphs 7 and 8 also show us negative estimates which are very unrealistic. This is not a surprise because when we look at the underlying functions of LYFCNR and LNFCYR in graphs 3 and 4 we observe that they not evenly distributed. With all this, we are not in position to say that one algorithm is better than the other.

For 2D extrapolation, we can see in tables 29 and 30, Biquadratic seems to have better estimates for APT and MPT. This is also true when we compare with the results from graphs 9 and 10. When we look at tables 31 and 32, we observe that the lower order algorithms seem to be better for LYFCNR and LNFCYR, but when we compare with results from the graphs 11 and 12, we also observe that most estimates for the different algorithms are out of range. Just like for 2D interpolation, we are also not in position to say that one algorithm is better than the other.

For 3D interpolation, tables 33 and 34, we can see that the lower order algorithms are relatively better estimators as compared to higher orders. When we also compare with the results from the graphs 13 and 14, we also observe the same pattern. In tables 35 and 36, we can see that the lower order algorithms seem to have better estimates for LYFCNR and LNFCYR. When we again look at the graphs 15 and 16, we also observe that the lower order algorithms are relatively better estimators much as they in some cases also give unrealistic results.

For 3D extrapolation, tables 37 and 39, Triquadratic seems to have better estimates for APT and MPT. Comparing with the results from the graphs 17 and 18, we also see that Triquadratic in still a better estimator. In tables 39 and 40, the lower order extrapolation algorithms seem to be better for LYFCNR and LNFCYR. When we also compare with

the results from graphs 19 and 20, we also observe that the lower order algorithms seem to have better estimators much as they in some cases also give unrealistic results.

6.3.1 Comparison of closer and distant data points

Tables 41 and 42 shows us a comparison of closer and distant data points in estimating unknown points for APT and MPT with 4-term bilinear interpolation. The data points used in calculating the coefficients of equation 3.3 are located at the rectangular coordinates of below:

- Stack-Size (5 – 20) and Min Gradient (2 - 10): for distant data points
- Stack-Size (5 – 10) and Min Gradient (2 - 5): for closer data points.

From the tables we can see that for the same point being interpolated, closer data points produce better estimates for both APT and MPT.

7 Summary and Conclusion

This paper presented different interpolation/extrapolation algorithms for estimating values in 2-dimensions and 3-dimensions. The problem stated in chapter 3 has been solved: as shown in chapters 4 through 6, algorithms for estimating complex dependencies have been developed.

The results presented in chapter 6 above do not show superiority of any algorithm. In this case, we do not have enough evidence to accept or reject the hypothesis stated at the end of chapter 4.

In a situation like the one in this paper, where there is no dominating algorithm in producing better estimates, lower order algorithms like Level plane and Bilinear for 2D and 8-term Trilinear and 9-term Trilinear for 3D, seems a well thought alternative. The advantage of the lower order over the higher order is that they use fewer data points in estimating an unknown and they also have the lowest execution time since they do not perform many computations.

From the results presented in chapter 6, we can conclude that the different algorithms presented in chapter 4 can be reliable enough in estimating dependencies that have a directional association with respect to the value bindings like APT and MPT but not very reliable for those those have abstract association with respect to the value bindings like LYFCNR and LNFCYR.

We however would like to state that the algorithms were designed are not entirely complete. The search algorithm was only designed for the Level plane and Bilinear for 2D and 8-term and 9-term Trilinear but though not implemented on COVAMOF data because of the small data set used. The search algorithm helps in finding the most appropriate interpolating matrix with data points closest to the point being estimated. When we look back in section 6.3.1, we can see that data points that are closer to the

point being estimated produce better estimates. The search algorithm for quadratic and cubic were not designed because of time.

8 References

[1] M. D McIlroy, "Mass Produced Software Components", in *Proceedings of the NATO Software Engineering Conference,* Garmisch, Germany, pp. 138-155, 1968.

[2] D. L. Parnas, "On the Design and Development of Program Families" *IEEE Transactions on Software Engineering,* Vol. 2, No. 1, pp. 1-9, 1976.

[3] J. Bosch, *Design & Use of Software Architectures: Adopting and Evolving a Product-Line Approach,* Addison-Wesley, 2000.

[4] Pohl, K.; Böckle, G.; van der Linden, F.: Software Product Line Engineering - Foundations, Principles, and Techniques. Heidelberg: Springer-Verlag, 2005

[5] Raatikainen, M., Soininen, T., Männistö, T., Mattila, A.: A Case Study of Two Configurable Software Product Families. In: *Proceedings of the 5th International Workshop on Product Family Engineering (PFE-5). Lecture Notes in Computer Science 3014.* 2004.

[6] Raatikainen, M., Soininen, T., Männistö, T., Mattila, A.: Characterizing Configurable Software Product Families and Their Derivation. Software Process: Improvement and Practices, *to appear.* 2005.

[7] Clements, P. C. and Northrop, L.: Software Product Lines - Practices and Patterns. Addison-Wesley, Boston (MA). 2001.

[8] Clements, P., Northrop, L.: Salion, inc.: A Software Product Line Case Study. CMU/SEI-2002-TR-038 2002.

[9] M. Sinnema, S. Deelstra, J. Nijhuis, J. Bosch, Modeling Dependencies in Product Families with COVAMOF, Proceedings of the 13th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2006), March 2006.

[10] I. Jacobson, M. Griss, P. Jonsson: Software Reuse. Architecture, Process and Organization for Business Success. Addison-Wesley, ISBN: 0-201-92476-5, 1997.

[11] M. Sinnema, S. Deelstra, J. Nijhuis, J. Bosch, "COVAMOF: A Framework for Modeling Variability in Software Product Families", *Proceedings of the Third Software Product Line Conference (SPLC 2004), Springer Verlag Lecture Notes on Computer Science Vol. 3154 (LNCS 3154),* pp. 197-213, August 2004.

[12] M. Jaring and J. Bosch: "Variability Dependencies in Product Family Engineering",

in *Proceedings of the Fifth International Workshop on Software Product-Family Engineering,* Siena, Italy, pp. 81-98, 2003

[13] M. Jaring. *Variability Engineering as an integral part of the Software Product Family Development Process.* PhD thesis, University of Groningen. 2005.

[14] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in Fortran: The Art of Scientific Computing*, Cambridge University Press, 1992

[15] M. Sinnema, S. Deelstra, P. Hoekstra, The COVAMOF Derivation Process, Proceedings of the 9th International Conference on Software Reuse (ICSR 2006), June 2006.

[16] Microsoft Visual Studio .NET, http://msdn.microsoft.com/vstudio.

[17] R. L. Burden, J. D. Faires, "*Numerical Analysis*", Youngstown State University, 2001.

[18] L. Cox, H. S. Delugach, D. Skipper, "Dependency Analysis Using Conceptual Graphs", Retrieved January 2006, from, http://www.cs.uah.edu/~delugach/Papers/CoxDelugachSkipper2001.pdf

[19] S. Deelstra, M. Sinnema, J. Bosch, Product Derivation in Software Product Families; A Case Study, Journal of Systems and Software, Vol 74/2 pp. 173-194, January 2005.

[20] D. Kidner, M. Dorey, D. Smith, "What's the point? Interpolation and extrapolation with a regular grid", Retrieved May 2006, from, http://www.geovista.psu.edu/sites/geocomp99/Gc99/082/gc_082.htm

[21] P. Bourke, "Trilinear Interpolation" Retrieved May 2006, from, http://astronomy.swin.edu.au/~pbourke/other/trilinear/

[22] W.S. Russell, 1995, Polynomial interpolation schemes for internal derivative distributions on structured grids. Applied Numerical Mathematics, 17, 129–171.

[23] F. Lekien, J. Marsden, "Tricubic interpolation in three dimensions", International Journal for Numerical Methods in Engineering, Vol 63 pp. 455-471, March 2005

[24] http://en.wikipedia.org/wiki/Extrapolation.

[26] L. L. Schumaker, Fitting surfaces to scattered data. In Approximation Theory II, edited by G. G. Lorentz, C. K. Chui and L. L. Schumaker (N.Y.: Academic Press), pp. 203-268.

9 Appendix A

9.1 2D interpolation results

SS	MG	APT	MPT	LNFCYR	LYFCNR
6	2	0.458714	1.093	0.006757	0.023715
6	9	0.488002	1.157	0.016892	0.023715
12	9	0.689029	1.703	0.023649	0.025692

Table 3. COVAMOF Experiment Results

SS	MG	APT	MPT	LNFCYR	LYFCNR
6	2	0.444752	1.050	0.005928	0.023715
6	9	0.449725	0.977	0.033597	0.030180
12	9	0.654776	1.653	0.045455	0.022072

Table 4. Level plane Interpolation

SS	MG	APT	MPT	LNFCYR	LYFCNR
6	2	0.444649	1.050	0.007510	-0.001350
6	9	0.449862	0.977	0.031489	0.030180
12	9	0.655735	1.653	0.030698	0.022072

Table 5. Bilinear Interpolation

SS	MG	APT	MPT	LNFCYR	LYFCNR
6	2	0.435218	1.052	-0.013202	-0.018782
6	9	0.444827	0.975	0.026640	0.026937
12	9	0.668938	1.682	0.028854	0.028153

Table 6. 8-term Biquadratic Interpolation

SS	MG	APT	MPT	LNFCYR	LYFCNR
6	2	0.437402	1.056	-0.012727	-0.017160
6	9	0.445313	0.976	0.026746	0.027297
12	9	0.668938	1.679	0.028485	0.026892

Table 7. 9-term Biquadratic Interpolation

SS	MG	APT	MPT	LNFCYR	LYFCNR
6	2	0.500735	1.270	0.019637	0.009514
6	9	0.443724	0.976	0.027652	0.028953
12	9	0.668938	1.689	0.028731	0.025495

Table 8. 16-term Bicubic Interpolation

9.2 2D Extrapolation results

SS	MG	APT	MPT	LNFCYR	LYFCNR
12	12	0.700757	1.657	0.067568	0.017787
3	12	0.351584	0.656	0.067568	0.023715
3	9	0.358344	0.688	0.020270	0.019763

Table 9. COVAMOF Experiment Results

SS	MG	APT	MPT	LNFCYR	LYFCNR
12	12	0.656907	1.622	0.057313	0.035585
3	12	0.349330	0.608	0.039526	0.047747
3	9	0.347199	0.639	0.027668	0.034234

Table 10. 3-term Level Plane Extrapolation

SS	MG	APT	MPT	LNFCYR	LYFCNR
12	12	0.658586	1.622	0.031487	0.035585
3	12	0.348850	0.608	0.046905	0.047747
3	9	0.346925	0.639	0.031884	0.034234

Table 11. 4-term Bilinear Extrapolation

SS	MG	APT	MPT	LNFCYR	LYFCNR
12	12	0.680223	1.704	0.023007	0.076262
6	9	0.342106	0.615	0.001897	-0.013737

3	9	.356073	0.658	0.024822	0.025721

Table 12. 8-term Biquadratic Extrapolation

SS	MG	APT	MPT	LNFCYR	LYFCNR
12	12	0.668330	1.684	0.020423	0.067433
6	9	0.330213	0.595	-0.000688	-0.022566
3	9	0.354374	0.655	0.024453	0.024460

Table 13. 9-term Biquadratic Extrapolation

SS	MG	APT	MPT	LNFCYR	LYFCNR
12	12	0.661424	1.635	0.017503	0.069577
6	9	0.253640	0.381	-0.018830	-0.055137
3	9	0.375368	0.694	0.025901	0.021814

Table 14. 16-term Bicubic Extrapolation

9.3 3D interpolation results

SS	MG	AMS	APT	MPT	LNFCYR	LYFCNR
6	2	3	0.458714	1.093	0.006757	0.023715
6	9	3	0.488002	1.157	0.016892	0.023715
12	9	3	0.689029	1.703	0.023649	0.025692

Table 15. COVAMOF Experiment Results

SS	MG	AMS	APT	MPT	LNFCYR	LYFCNR
6	2	3	0.445363	1.063	0.013043	0.007568
6	9	3	0.448910	0.985	0.021344	0.022230
12	9	3	0.649073	1.681	0.026086	0.025879

Table 16. 7-term Trilinear Interpolation

SS	MG	AMS	APT	MPT	LNFCYR	LYFCNR
6	2	3	0.444649	1.061	0.013162	0.007365
6	9	3	0.449862	0.988	0.021186	0.022500
12	9	3	0.655735	1.707	0.024980	0.027771

Table 17. 8-term Trilinear Interpolation

SS	MG	AMS	APT	MPT	LNFCYR	LYFCNR
6	2	3	0.437402	1.056	-0.012727	-0.017160
6	9	3	0.445313	0.976	0.026746	0.027297
12	9	3	0.667239	1.679	0.028485	0.026892

Table 18. 27-term Triquadratic Interpolation

SS	MG	AMS	APT	MPT	LNFCYR	LYFCNR
6	2	3	0.500735	1.270	0.019637	0.009514
6	9	3	0.443724	0.976	0.027652	0.028953
12	9	3	0.671613	1.689	0.028731	0.025495

Table 19. 64-term Tricubic Interpolation

9.4 3D extrapolation results

SS	MG	AMS	APT	MPT	LNFCYR	LYFCNR
12	12	3	0.700757	1.657	0.067568	0.017787
3	12	3	0.351584	0.656	0.067568	0.023715
3	9	3	0.358344	0.688	0.020270	0.019763

Table 20. COVAMOF Experiment Results

SS	MG	AMS	APT	MPT	LNFCYR	LYFCNR
12	12	3	0.646927	1.615	0.029643	0.033379
3	12	3	0.352181	0.619	0.022530	0.026081
13	9	3	0.348828	0.636	0.018972	0.020406

Table 21. 7-term Trilinear Extrapolation

SS	MG	AMS	APT	MPT	LNFCYR	LYFCNR
12	12	3	0.658586	1.659	0.027708	0.036689
3	12	3	0.348850	0.606	0.023083	0.025135
3	9	3	0.346925	0.629	0.019288	0.019865

Table 22. 8-term Trilinear Extrapolation

SS	MG	AMS	APT	MPT	LNFCYR	LYFCNR
12	12	3	0.668330	1.684	0.020423	0.067433

3	12	3	0.330213	0.595	-0.000688	-0.022566
3	9	3	0.354374	0.655	0.024453	0.024460

Table 23. 27-term Triquadratic Extrapolation

SS	MG	AMS	APT	MPT	LNFCYR	LYFCNR
12	12	3	0.661424	1.635	0.017503	0.069577
3	12	3	0.253640	0.381	-0.018830	-0.055137
3	9	3	0.375368	0.694	0.025901	0.021814

Table 24. 64-term Tricubic Extrapolation

9.5 2D interpolation RMSE

Average Processing Time	
Interpolation algorithm	**Rms. Error (6-observations)**
3-term level plane	0.0437
4-term Bilinear	0.0438
8-term Biquadratic	0.0391
9-term Biquadratic	0.0414
16-term Bicubic	0.0378

Table 25. APT rms. Error for 2D Interpolation algorithms

Maximum Processing Time	
Interpolation algorithm	**Rms. Error (6-observations)**
3-term level plane	0.2004
4-term Bilinear	0.2004
8-term Biquadratic	0.1677
9-term Biquadratic	0.1717
16-term Bicubic	0.1466

Table 26. MPT rms. Error for 2D Interpolation algorithms

LabelNoFCYes Rate	
Interpolation algorithm	**Rms. Error (6-observations)**

3-term level plane	0.0122
4-term Bilinear	0.0116
8-term Biquadratic	0.0102
9-term Biquadratic	0.0097
16-term Bicubic	0.0104

Table 27. LNFCYR rms. Error for 2D Interpolation algorithms

LabelYesFCNo Rate	
Interpolation algorithm	**Rms. Error (6-observations)**
3-term level plane	0.0223
4-term Bilinear	0.0223
8-term Biquadratic	0.0213
9-term Biquadratic	0.0208
16-term Bicubic	0.0140

Table 28. LYFCNR rms. Error for 2D Interpolation algorithms

9.6 2D extrapolation RMSE

Average Processing Time	
Extrapolation algorithm	**Rms. Error (9-observations)**
3-term level plane	0.1199
4-term Bilinear	0.1193
8-term Biquadratic	0.0852
9-term Biquadratic	0.0619
16-term Bicubic	0.1668

Table 29. APT rms. Error for 2D Extrapolation algorithms

Maximum Processing Time	
Extrapolation algorithm	**Rms. Error (9-observations)**
3-term level plane	0.8437
4-term Bilinear	0.8437

8-term Biquadratic	0.4955
9-term Biquadratic	0.5765
16-term Bicubic	0.8532

Table 30. MPT rms. Error for 2D Extrapolation algorithms

LabelNoFCYes Rate	
Extrapolation algorithm	**Rms. Error (9-observations)**
3-term level plane	0.0224
4-term Bilinear	0.0345
8-term Biquadratic	0.0375
9-term Biquadratic	0.0461
16-term Bicubic	0.0506

Table 31`. LNFCYR rms. Error for 2D Extrapolation algorithms

LabelYesFCNo Rate	
Extrapolation algorithm	**Rms. Error (9-observations)**
3-term level plane	0.0270
4-term Bilinear	0.0270
8-term Biquadratic	0.1023
9-term Biquadratic	0.0678
16-term Bicubic	0.0644

Table 32. LYFCNR rms. Error for 2D Extrapolation algorithms

9.7 3D interpolation RMSE

Average Processing Time	
Interpolation algorithm	**Rms. Error (12-observations)**
7-term Trilinear	0.0366
8-term Trilinear	0.0371
27-term Triquadratic	0.0347
64-term Triquadratic	0.0418

Table 33. APT rms. Error for 3D Interpolation algorithms

Maximum Processing Time	
Interpolation algorithm	**Rms. Error (12-observations)**
7-term Trilinear	0.1950
8-term Trilinear	0.1980
27-term Triquadratic	0.2357
64-term Triquadratic	0.2966

Table 34. MPT rms. Error for 3D Interpolation algorithms

LabelNoFCYes Rate	
Interpolation algorithm	**Rms. Error (12-observations)**
7-term Trilinear	0.0092
8-term Trilinear	0.0093
27-term Triquadratic	0.0128
64-term Triquadratic	0.0140

Table 35. LNFCYR rms. Error for 3D Interpolation algorithms

LabelYesFCNo Rate	
Interpolation algorithm	**Rms. Error (12-observations)**
7-term Trilinear	0.0124
8-term Trilinear	0.0126
27-term Triquadratic	0.0210
64-term Triquadratic	0.0152

Table 36. LYFCNR rms. Error for 3D Interpolation algorithms

9.8 3D extrapolation RMSE

Average Processing Time	
Extrapolation algorithm	**Rms. Error (17-observations)**
7-term Trilinear	0.0920

8-term Trilinear	0.0886
27-term Triquadratic	0.0476
64-term Triquadratic	0.1204

Table 37. APT rms. Error for 3D Extrapolation algorithms

Maximum Processing Time	
Extrapolation algorithm	**Rms. Error (17-observations)**
7-term Trilinear	0.5708
8-term Trilinear	0.5713
27-term Triquadratic	0.4445
64-term Triquadratic	1.1825

Table 38. MPT rms. Error for 3D Extrapolation algorithms

LabelNoFCYes Rate	
Extrapolation algorithm	**Rms. Error (17-observations)**
7-term Trilinear	0.0253
8-term Trilinear	0.0258
27-term Triquadratic	0.0486
64-term Triquadratic	0.0512

Table 39. LNFCYR rms. Error for 3D Extrapolation algorithms

LabelYesFCNo Rate	
Extrapolation algorithm	**Rms. Error (17-observations)**
7-term Trilinear	0.0138
8-term Trilinear	0.0153
27-term Triquadratic	0.0769
64-term Triquadratic	0.0531

Table 40. LYFCNR rms. Error for 3D Extrapolation algorithms

APT			
Value Bindings (SS,MG)	Experiment	Est(closer data points)	Est (distant data points)
6,2	0.458714	0.516677	0.529650
6,4	0.471662	0.470148	0.509505

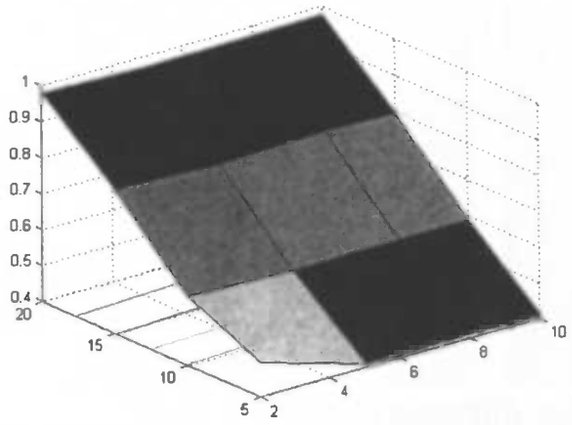
Table 41. Comparison of closer data points and distant data points in estimating an APT.

MPT			
Value Bindings (SS,MG)	Experiment	Est(closer data points)	Est (distant data points)
6,2	1.093	1.316	1.355
6,4	1.172	1.118	1.260

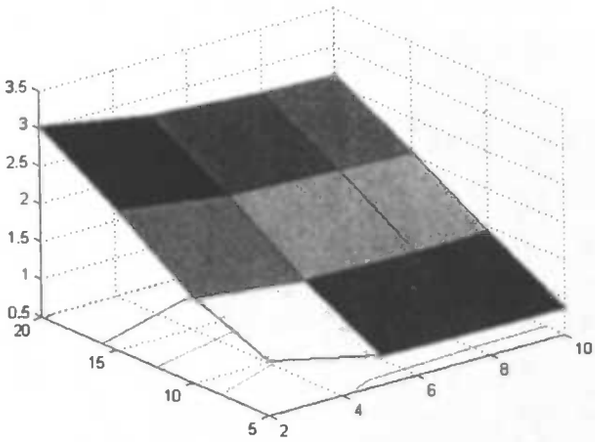
Table 42. Comparison of closer data points and distant data points in estimating MPT.

10 Appendix B

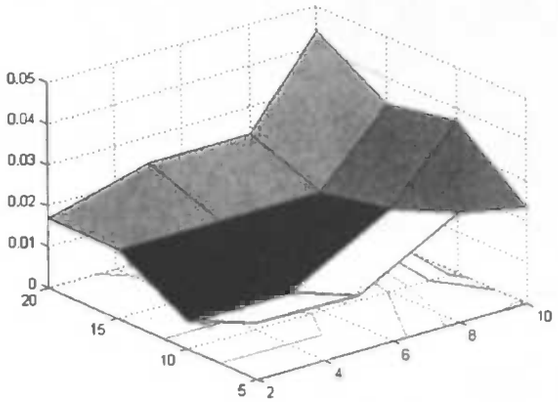
10.1 Graphs for different underlying function in 2D



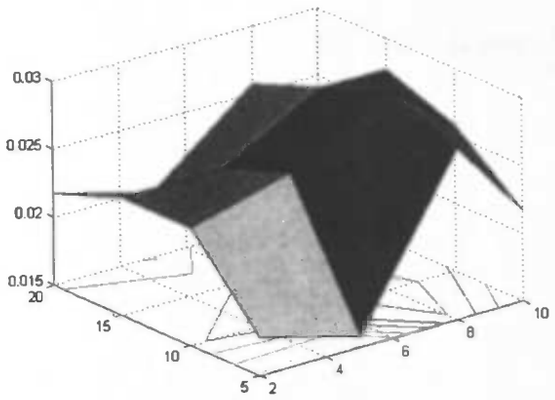
Graph 1. 2D Average processing time underlying function.



Graph 2. 2D Maximum processing Time underlying function.

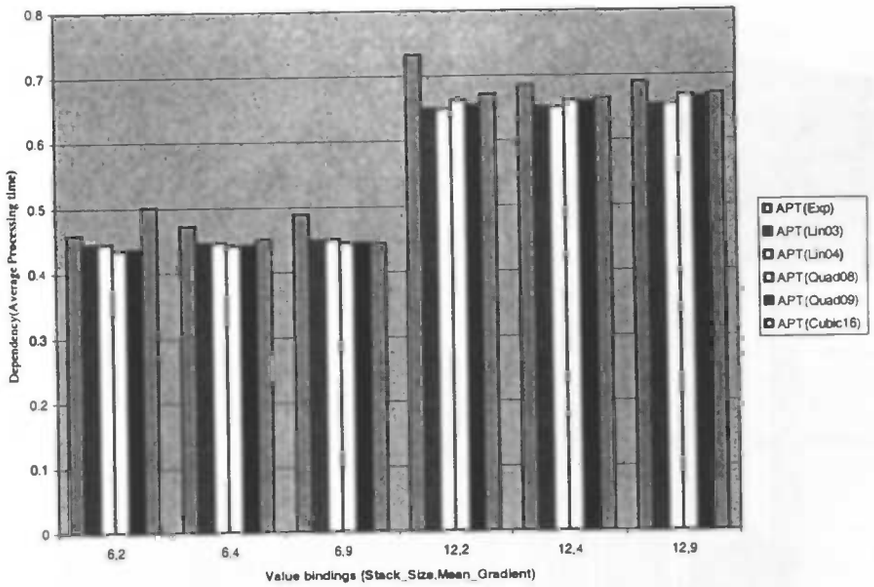


Graph 3. 2D LabelNoFCYes Rate Underlying function.

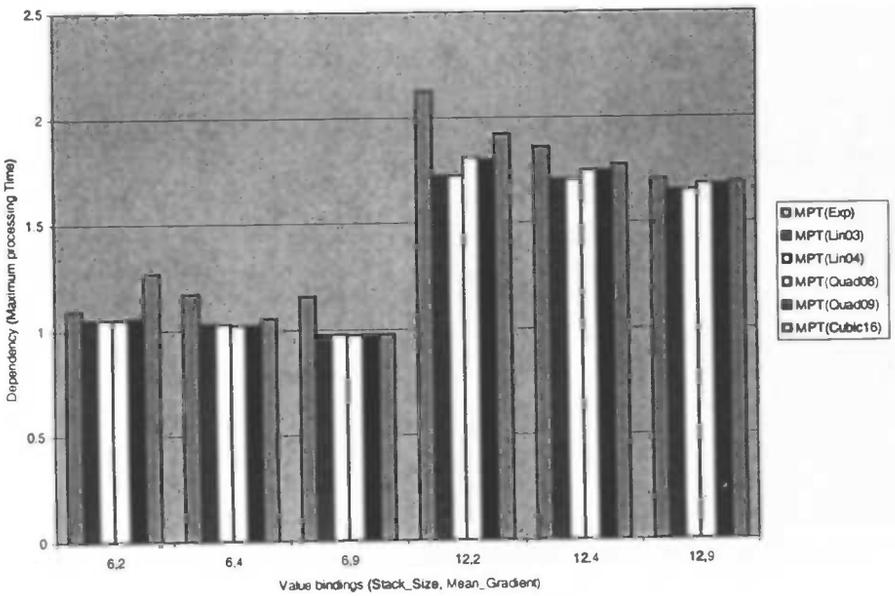


Graph 4. 2D LabelYesFCNo Rate Underlying function.

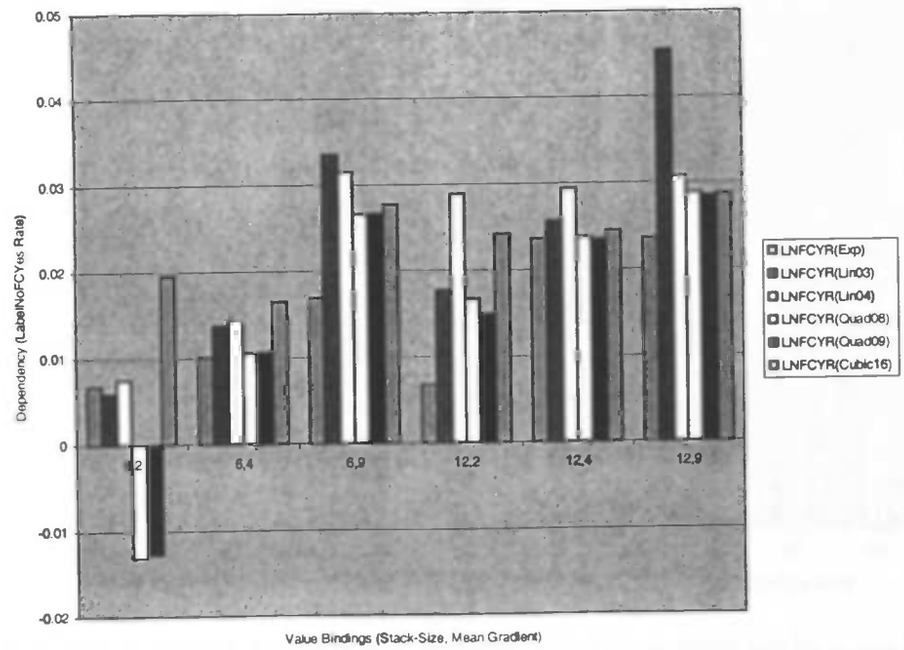
10.2 Graphs for the comparison of the different algorithms in 2D



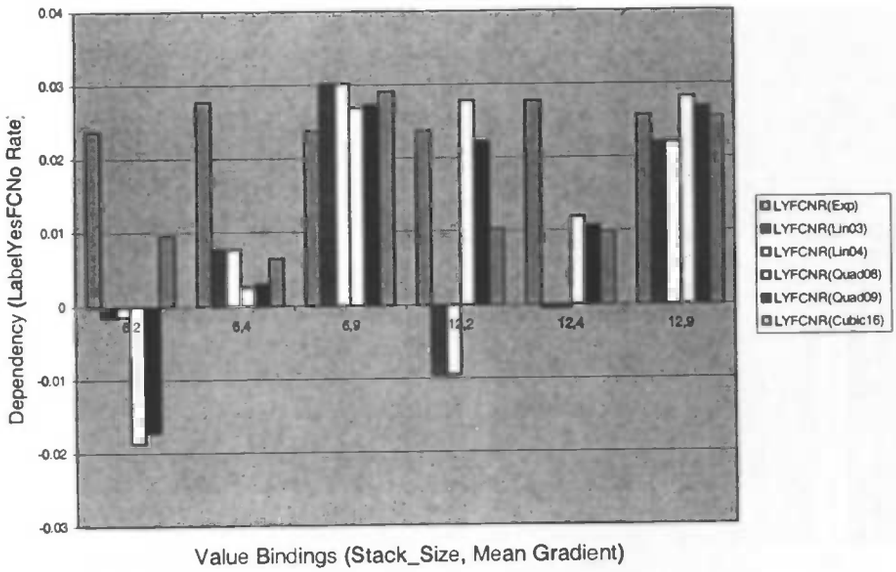
Graph 5. Comparison of the different 2D algoritms in interpolating Average processing Time.



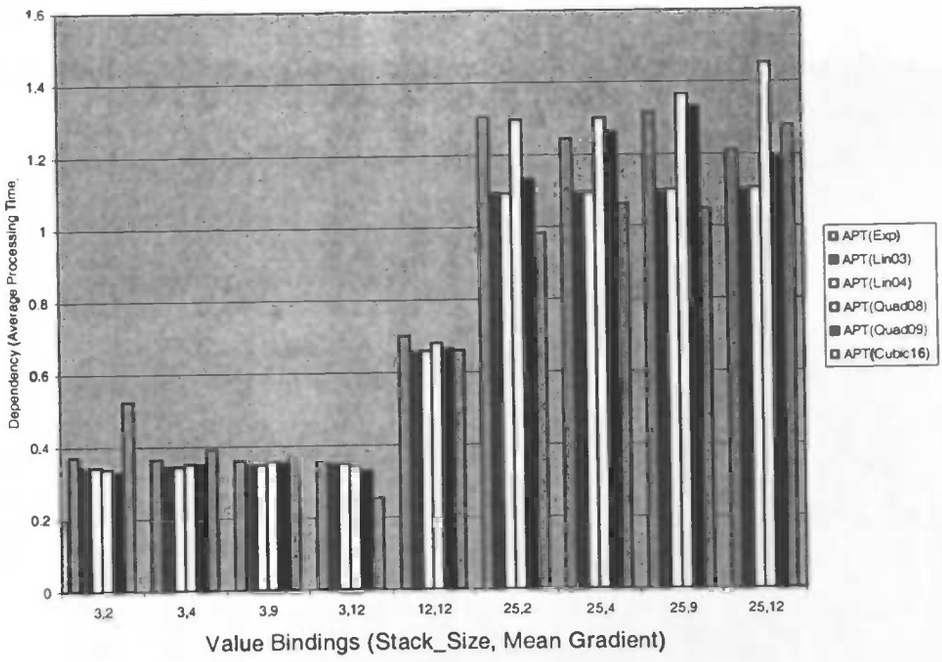
Graph 6. Comparison of the different 2D algorithms in interpolating Maximum Processing Time



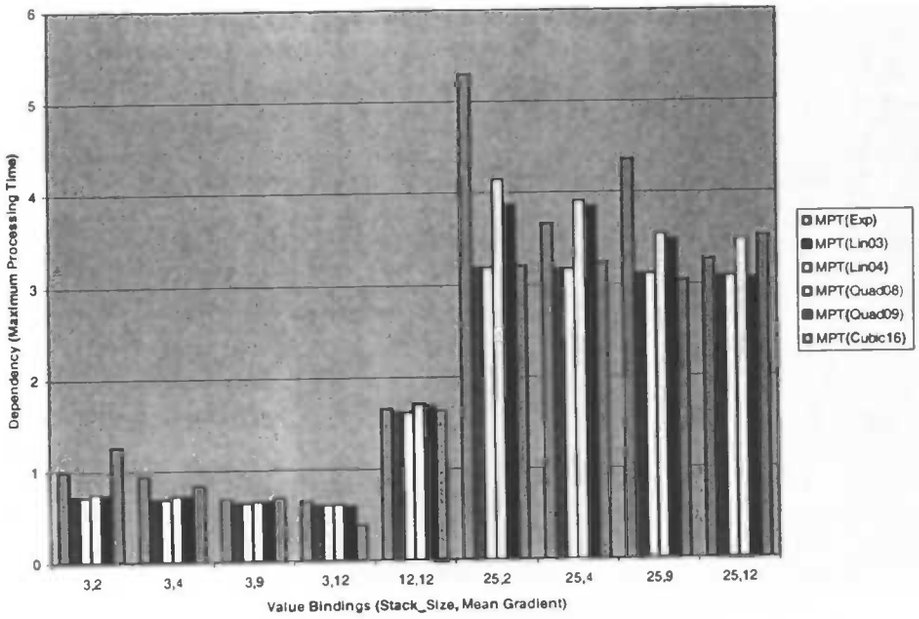
Graph 7. Comparison of the different 2D algorithms in interpolating LabelNoFCYes Rate.



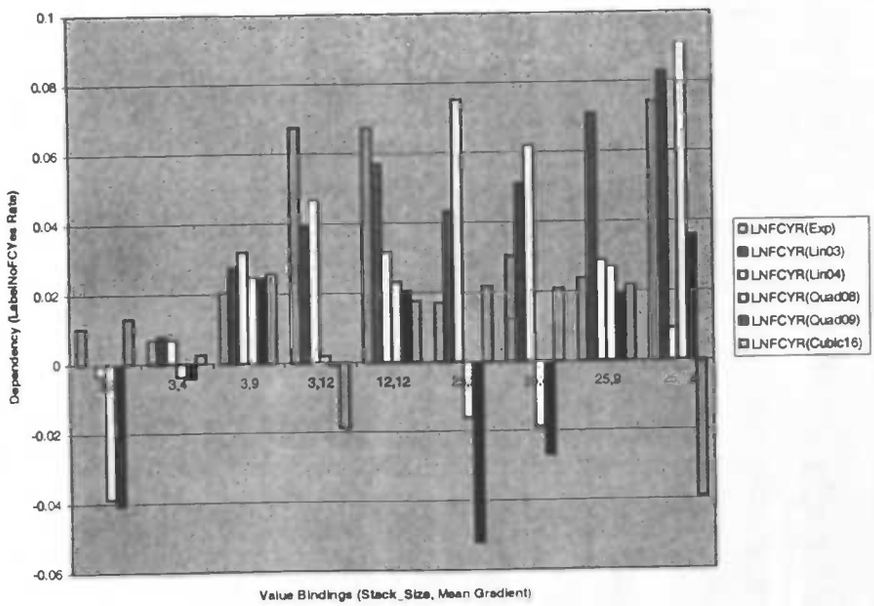
Graph 8. Comparison of the different 2D algorithms in interpolating LabelYesFCNo Rate



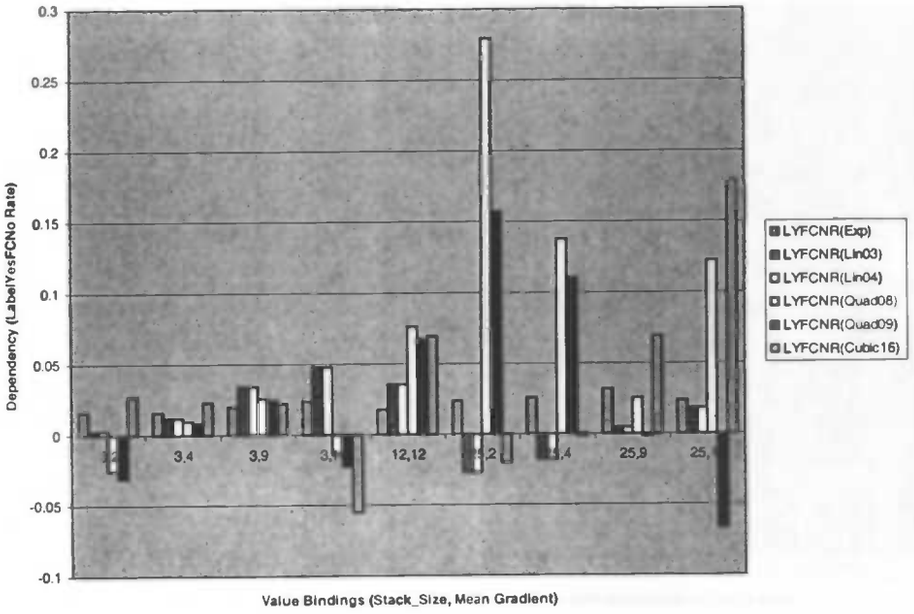
Graph 9. Comparison of the different 2D algorithms in extrapolating Average processing Time.



Graph 10. Comparison of the different 2D algorithms in extrapolating Maximum Processing Time.

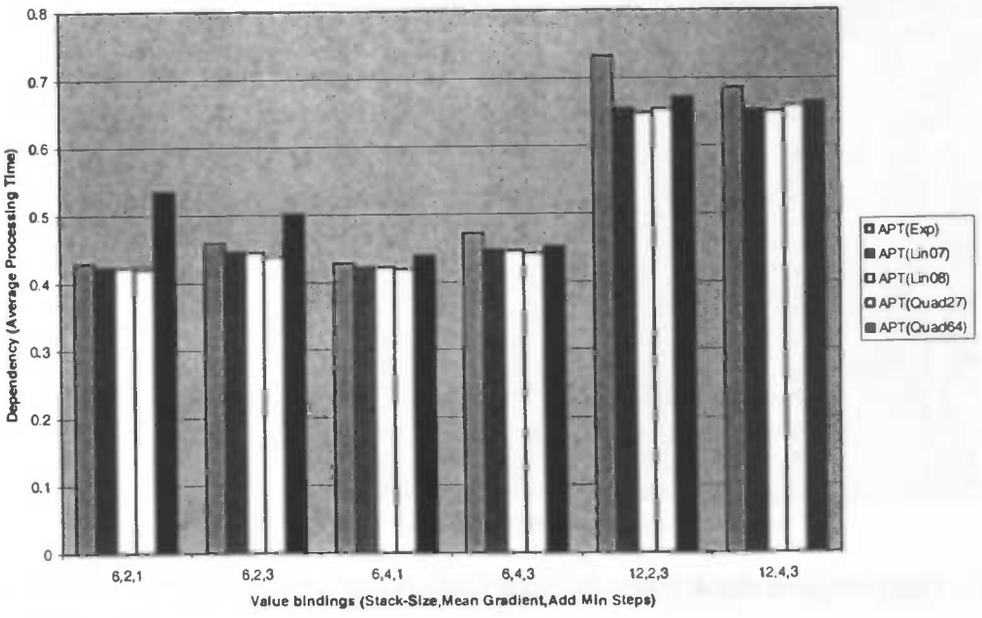


Graph 11. Comparison of the different 2D algorithms in extrapolating LabelNoFCYes Rate.

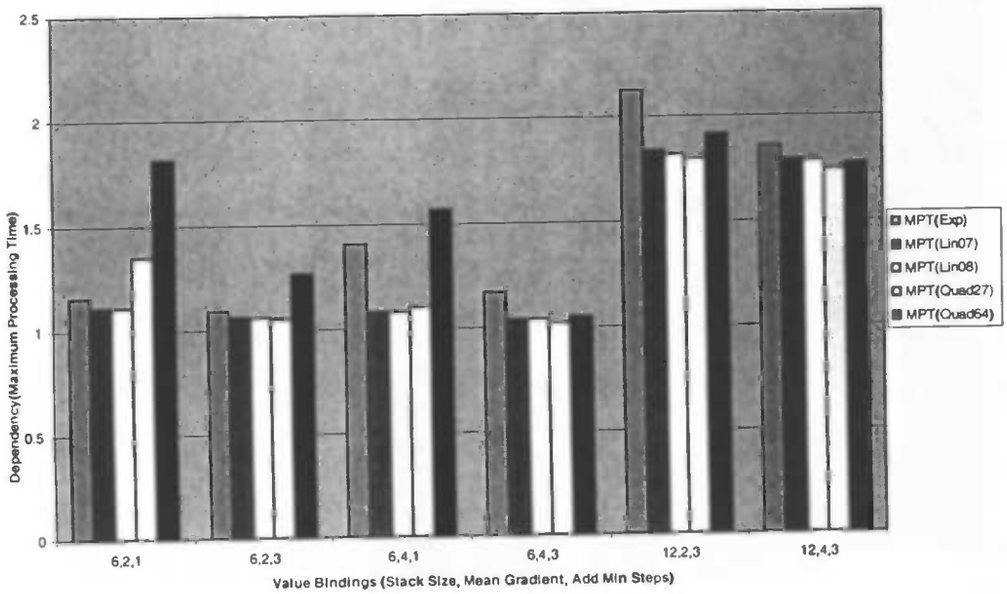


Graph 12. Comparison of the different 2D algorithms in extrapolating LabelYesFCNo Rate.

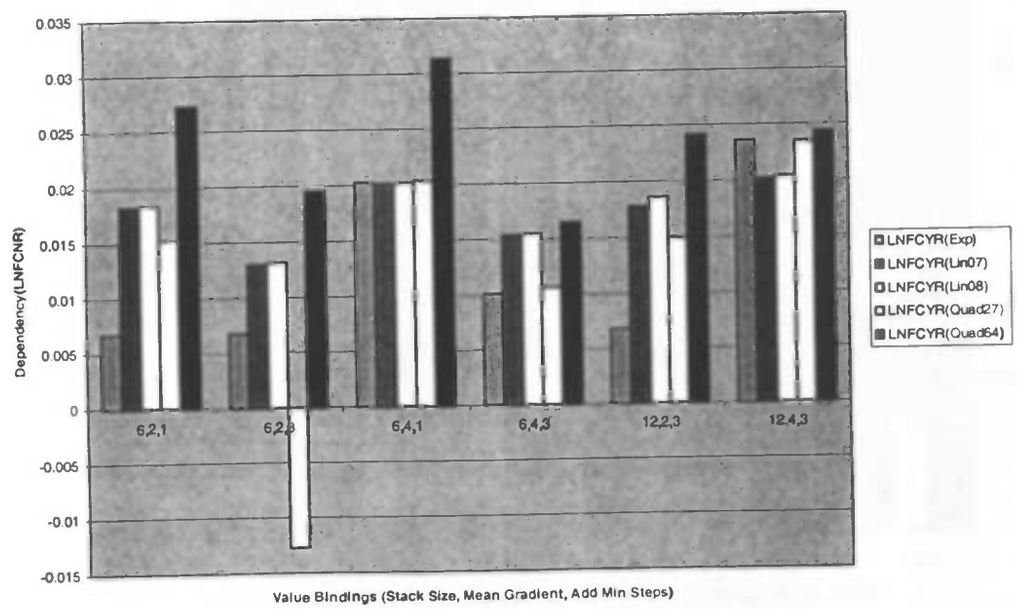
10.3 Graphs for the comparison of the different 3D algorithms



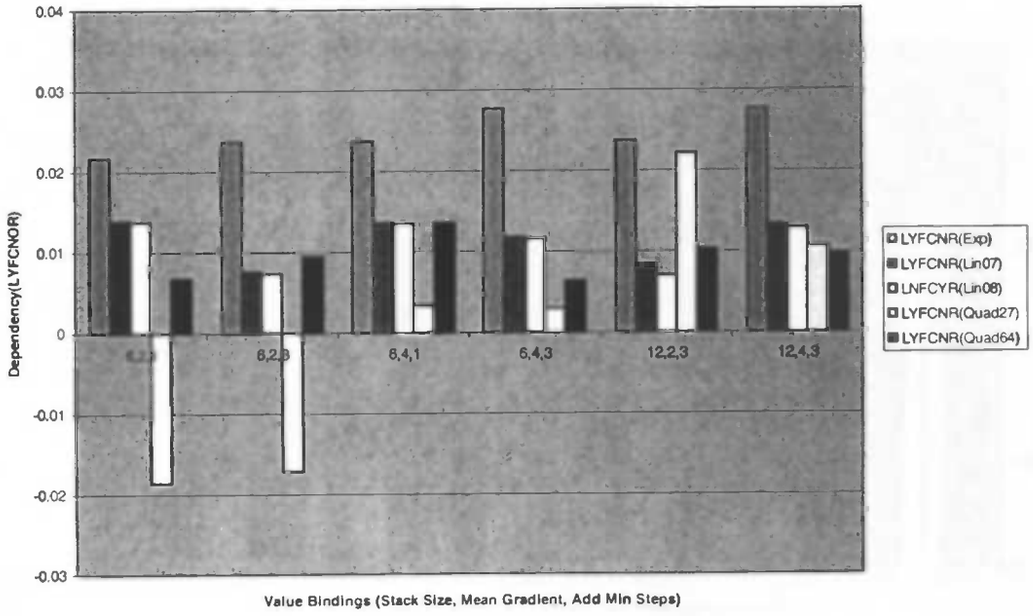
Graph 13. Comparison of the different 3D algorithms in interpolating Average Processing Time.



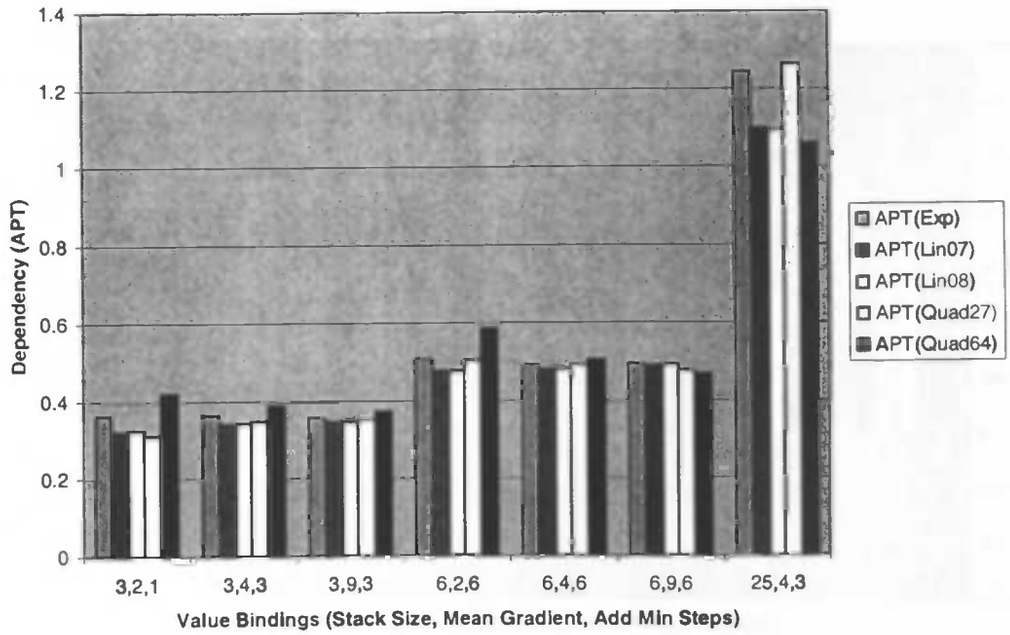
Graph 14. Comparison of the different 3D algorithms in interpolating Maximum Processing Time.



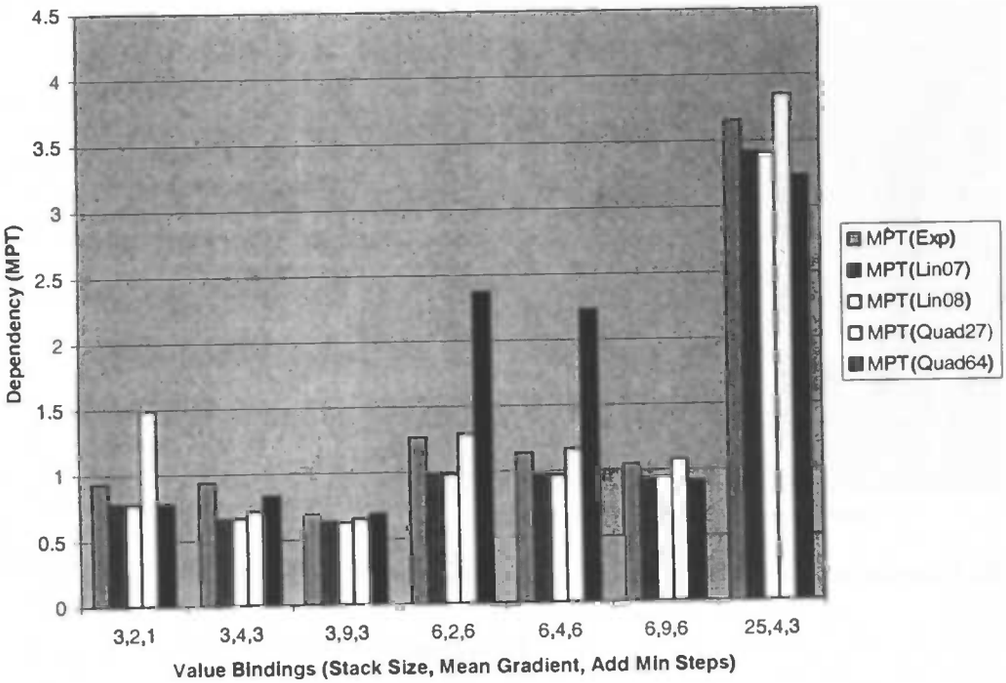
Graph 15. Comparison of the different 3D algorithms in interpolating LabelNoFCYes Rate



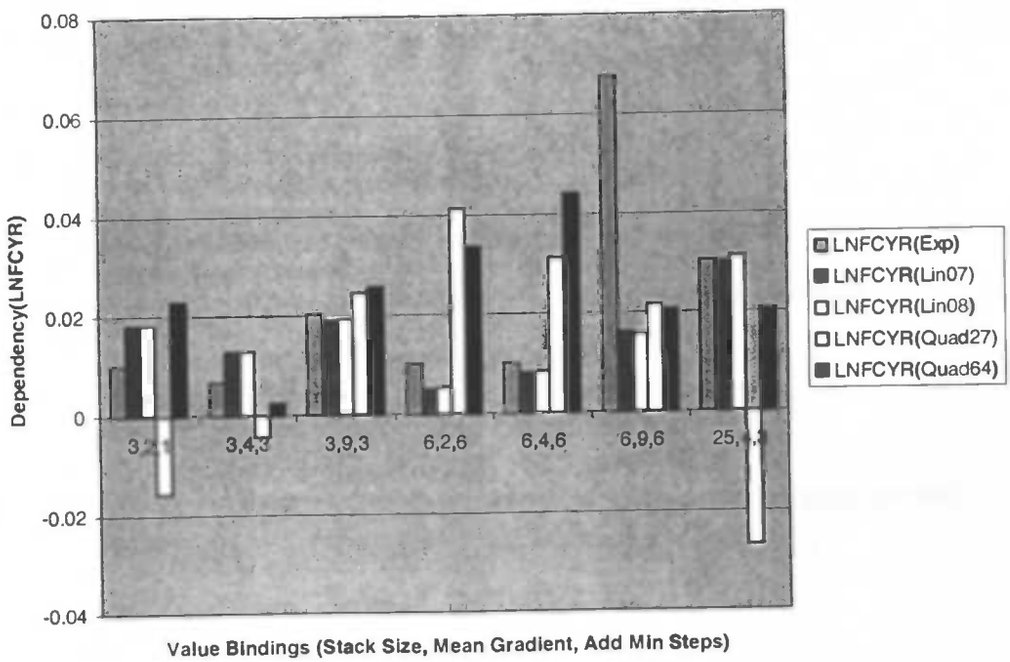
Graph 16. Comparison of the different 3D algorithms in interpolating LabelYesFCNo Rate.



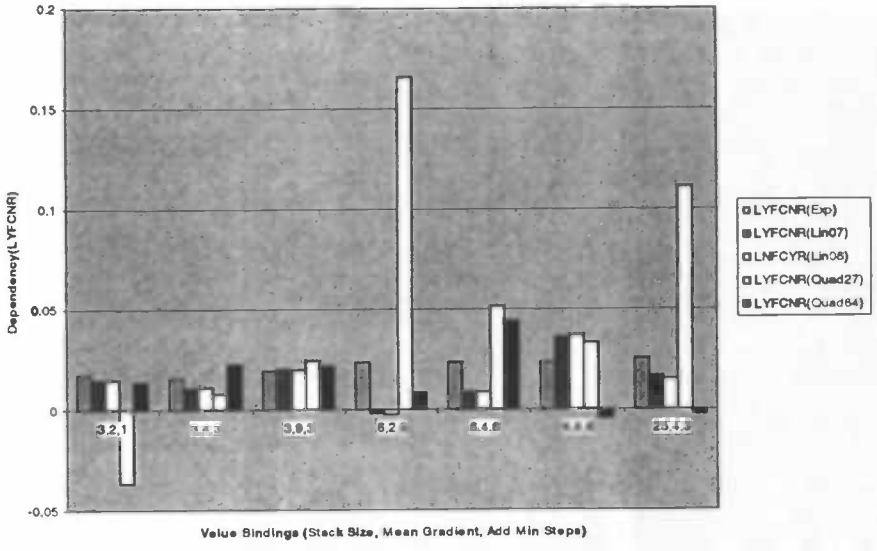
Graph 17. Comparison of the different 3D algorithms in extrapolating Average Processing Time.



Graph 18 Comparison of the different 3D algorithms in extrapolating Maximum Processing Time.



Graph 19. Comparison of the different 3D algorithms in extrapolating LabelNoFCYes Rate.



Graph 20. Comparison of the different 3D algorithms in extrapolating LabelYesFCNo Rate.

11 Appendix C

11.1 Function for parsing xml file

```
function [VA,Dep,Dim] = parseXMLDocument(filename)
%This function parses an xml file and returns the value bindings in array
%VA the dependency result in array Dep and the dimension of the dependency
% PARSEXML Convert XML file to a MATLAB structure.
try
  tree = xmlread(filename);
catch
  error('Failed to read XML file %s.',filename);
end
 % Recurse over child nodes. This could run into problems
% with very deeply nested trees.
try
  [VA,Dep,Dim] = parseChildNodes(tree);
catch
  error('Unable to parse XML file %s.',filename);
end

 function [VA,Dep,Dim] = parseChildNodes(node)
   n = 1;
   m = 1;
   nodes = node.getChildNodes;
   if nodes.hasChildNodes %nodes != null
      children = nodes.getChildNodes;
      if children.hasChildNodes
        child = children.item(0).getChildNodes;
        for i = 2:child.getLength-1
           childs = child.item(i-1);
           c = childs.getNodeName;
           attr = childs.getAttributes.item(0).getNodeValue;
           refData = child.item(i-1).getChildNodes;
           if refData.hasChildNodes
            for j = 2:2:refData.getLength-1
              refDatach = refData.item(j-1);
              attrRef(n) = str2num(refDatach.getAttributes.item(0).getNodeValue);
              n = n+1;
              valBind = refData.item(j-1).getChildNodes;
              if valBind.hasChildNodes
                 r = 0;
                 for k = 2:2:valBind.getLength-1
                    valBindch = valBind.item(k-1);
                    attrValBind(m) = str2num(valBindch.getAttributes.item(0).getNodeValue);
                    m = m + 1;
                    r = r + 1;
                 end
              end
            end
           end
        end
        Dep=attrRef;
        VA = attrValBind;
```

```
      Dim = r;
    end
  end
```

11.2 Function for 2D sorting

```
function [A, B, D] = sort2D(VA, Dep)
%This function takes in an array of vlue bindings VA and an array of
%dependencies Dep and returns sorted arrays A and B that are value bindings from
%variation point A and B respectively. It also returns a matrix of
%dependencies that correspond to rectangular coordinates of array A and B.
Z = Dep;
a = 1;
for i = 1:2:length(VA)
  X(a) = VA(i);
  Y(a) = VA(i+1);
  a = a + 1;
end

for i = 1:length(X)
   for j=1:length(X)-i
      if (X(j) > X(j+1) || (X(j) == X(j+1) && Y(j) > Y(j+1)))
         tempX = X(j);
         tempY = Y(j);
         tempZ = Z(j);

         X(j)=X(j+1);
         Y(j)=Y(j+1);
         Z(j)=Z(j+1);

         X(j+1)=tempX;
         Y(j+1)=tempY;
         Z(j+1)=tempZ;
      end
   end
end

n = 1;
B = [];
for i = 1:length(X)
   B = InsertValue(Y, B, i);
   if (i == 1 || X(i) ~= X(i-1))
      A(n) = X(i);
      n = n + 1;
   end
end

for i = 1:length(A)
   for j = 1:length(B)
      for n = 1:length(X)
         flag = 0;
        if A(i) == X(n) && B(j) == Y(n)
```

```
            D(i,j) = Z(n);
            flag = 1;
            break;
          end
        end
        if flag == 0
           D(i,j) = -100;
        end
    end
end
```

11.3 Function for calculating the coefficients of an equation

```
function a = coefficient(A)
%This function takes in an n by n+1 matrix produced A from a n equations.
%It then solves the equations simuliteneously and returns the coefficients
%of the equations

i = 1;
s = size(A);
n = s(1);
while i <= n-1;
   p = i;
   while abs(A(p,i)) <= 1.0e-20 && p < n
      p = p+1;
   end
   % STEP 3
   if p~=i
      for jj = 1:n+1
         C = A(i,jj);
         A(i,jj) = A(p,jj);
         A(p,jj) = C;
      end
   end
   % STEP 4
   jj = i+1;
   for j = jj:n
      % STEP 5
      m = A(j,i)/A(i,i);
      % STEP 6
      for k = jj:n+1
         A(j,k) = A(j,k) - m * A(i,k);
      end
      % Multiplier m could be saved in A(j,i).
      A(j,i) = 0;
   end
   i = i + 1;
end
A;
if abs(A(n,n)) <= 1.0e-20
  error('No Unique solution');
else
   a(n) = A(n,n+1) / A(n,n);
```

```
  for k = 1:n-1
     i = n-k;
     j = i+1;
     sum = 0;
     for kk = j:n
        sum = sum - A(i,kk) * a(kk);
     end
     a(i) = (A(i,n+1)+sum) / A(i,i);
  end
end
```

11.4 Function for estimating a point in 2D space

```
function Z = Estimate2D(filename,v1,v2)
%This function takes in an xml file with given data points and the
%coordinates of the point to be estimated and then returns the estimate.

[V1,V2,D] = sortValBinding2D(filename);

n = 0;
for i=1:length(V1)
   for j = 1:length(V2)
      if D(i,j) ~= -100
         D1(n+1) = D(i,j);
         V11(n+1) = V1(j);
         V22(n+1) = V2(i);
         n = n + 1;
      end
   end
end

for i = 1:n
   k2 = 0;
   j = 1;
   while k2 < length(V2)
      k1 = 0;
      while k1 < length(V1) && j < n+1
         A(i,j) = V11(i)^k1*V22(i)^k2;
         k1 = k1 + 1;
         j = j + 1;
      end
      if j == n+1;
         A(i,j) = D(i);
      end
      k2 = k2 + 1;
   end
end
A;
a = coefficient(A);

sum = 0;
i = 1;
k2 = 0;
while k2 < length(V2)
```

```
    k1 = 0;
    while k1 < length(V1) && i <= length(a)
        sum = sum + v1^k1*v2^k2*a(i);
        k1 = k1 + 1;
        i = i + 1;
    end
    k2 = k2 + 1;
end

Z = sum;
```