AN EXPERT SYSTEMS APPROACH TO COST ESTIMATION FOR STEEL STRUCTURES USING STOCHASTIC METHODS

by

ANDREW DONALD WATSON

B.A.Sc. (Civil) The University Of British Columbia 1989

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

Department Of Civil Engineering

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

November 1991

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of CIVIL ENGINEERING

The University of British Columbia
Vancouver, Canada

Date 20 SEPT 1991

Abstract

Cost information is important to all parties in the steel fabrication industry: owners; engineers; and fabricators. The industry is driven by economic forces that encourage the efficient use of material, labour and equipment making cost estimation a vital tool in every decision. Unfortunately, these same forces hinder accurate estimation by imposing economic constraints that reduce the amount of time, effort and money that can be committed to the process; therefore, the development of new and improved cost estimation methods is encouraged.

This thesis attempts to demonstrate the feasibility of applying expert systems technology as a means of estimating the fabrication cost of steel structures. It considers the historical factors limiting current methods of cost estimation and the strengths and weaknesses of computer technology in order to develop a new method of cost estimation that provides valuable information which is impractical to obtain using traditional methods. It introduces the concept of stochastic estimation which may be used to evaluate the variability of costs and assess accuracy. The new method is designed for efficient use of computer capabilities and is intended for development as a tool to aid engineers, estimators and fabricators.

Current methods of cost estimation are unsuitable for application in an expert system because they rely heavily on subjective judgement and are bound by constraints imposed by the burden of manual calculation. These constraints are no longer valid now that the industry has access to powerful personal computers that are capable of performing sophisticated calculations and manipulating large amounts of data. The

introduction of expert systems technology has provided an opportunity to examine traditional cost estimation methods and to propose new methods that are better suited for use in a computer-based environment.

The development of the new estimation method and its implementation as an expert system promises many benefits to the steel design and fabrication industries. It focuses the power of computer technology on the estimation problem. It provides greater access to accurate estimation, thereby improving the basis of important economic decisions. It can also be an important tool for the training of new cost estimators and allows the efforts of experienced personnel to be directed into more profitable areas. Moreover, an estimate derived by an expert system is consistent and unbiased making it better suited for comparison than one compiled by human estimators who may be influenced by personal preferences and external factors.

Table Of Contents

List Of Figures

Acknowledgements

I would like to take this opportunity to thank the Steel Structures Education Foundation (SSEF) for presenting me with the G.J. Jackson Fellowship and the Natural Science And Engineering Research Council (NSERC) for their scholarship which funded my research. In particular, I thank Hugh Krentz, Executive Director of the SSEF, and Dr. Siegfried F. Stiemer, my graduate advisor for their guidance on this project. I would also like to thank the engineers and fabricators who took the time to show me their facilities and discuss the principles and practice of cost estimation.

CHAPTER 1

INTRODUCTION AND JUSTIFICATION OF COMPUTER-AIDED COST ESTIMATION

A. Introduction

Cost estimation in the steel fabrication industry is the process of predicting the expense of producing the steel components of a structural system. This process can be conducted using a number of methods and philosophies, but is essentially a problem of modelling and data acquisition. Accurate cost estimation is essential to the owners, designers and fabricators of steel structures who rely on such estimates to arrange financing, minimize costs and produce competitively priced structures.

The purpose of this thesis is to create and develop a new method of cost estimation for implementation in a computer environment. The method is based on, but not limited by, current methods and concepts of cost estimation which are restricted by the practical capabilities of manual data transfer and calculation. The introduction of computer technology lifts this limitation and provides an opportunity to reevaluate the current basis of cost estimation and expand the toolkit of the cost estimator. The possibilities thus created are investigated and an innovative cost estimation method incorporating several new techniques is proposed.

The goal of this project is to assist in the development of an estimation programme which provides services equivalent to or better than those provided by an experienced steel fabrication cost estimator. This will be achieved using expert system programming

techniques which enable the computer to emulate the mental processes of a human estimator. The experience and judgement of the estimator are replaced by the gross computational power and data manipulation capabilities of computer technology [fig. 1]. In addition, new techniques including stochastic methods are introduced which provide information that can not be obtained practically using manual methods. When completed, this expert system can be a useful in the classroom, design office and fabrication shop.

B. Justification

Engineering, and civil engineering in particular, is the art of applying scientific principles to practical situations; therefore, the ultimate goal of every engineering thesis is to make some small improvement to a current method or idea. This implies that current methods are in some way unsatisfactory or that significant benefits are anticipated through the improvement of current methods, thereby justifying the research effort.

Cost estimation is an ideal candidate for research and improvement. Current methods of cost estimation are relatively simple and unsophisticated, yet the information provided by cost estimation is valuable and sought after. While the development of computer-based cost estimation methods requires a significant expenditure of time and effort, the development is justified because:

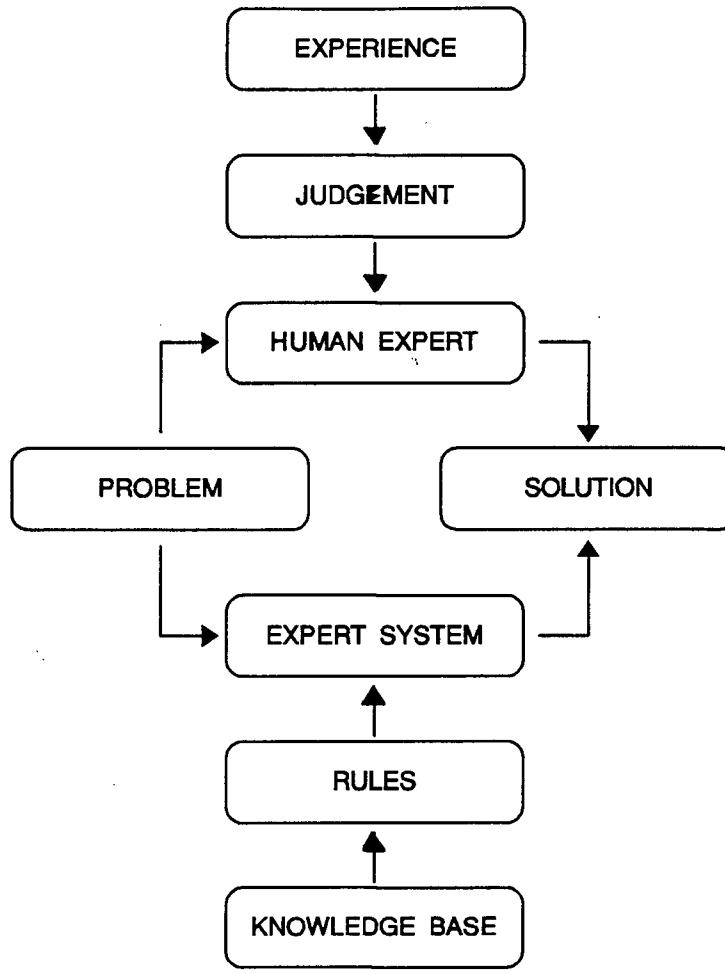


Figure 1: Parallel Tracks: Human Expert vs Expert System

a) current methods of cost estimation were developed for use before the introduction of computer technology and are no longer well suited for use in modern computer-oriented office environments;

b) low margins for contingencies and profit in the steel design and fabrication industry put a premium on the value of cost information;

c) the obvious economic benefits of fast and accurate cost estimation encourage the development, acceptance and use of estimation software;

d) the cost of steel fabrication is particularly sensitive to details and connections making it more difficult to produce accurate estimates using crude estimation techniques;

e) computer technology has matured to the point that automation of the task is feasible using available and affordable technology;

f) the basic operations of cost estimation, data transfer and calculation, can be performed faster and more accurately by computer than by a human estimator;

g) an interactive cost estimation programme would provide engineers with a tool for the design of more cost effective steel structures and a means of learning more about the fabrication process; and,

h) a fast and accurate estimation programme would provide fabricators with a means to examine the fabrication process and investigate alternate designs, thereby reducing their financial risk;

In economic terms, the development of estimation software for steel fabrication is justified as long as the value of information derived from the programme exceeds the costs of obtaining the information [fig. 2]. Economic justification applies economic pressure which encourages the development and acceptance of software by increasing the competitiveness of those designers and fabricators with access to it. Such software reduces the cost of steel fabrication by ensuring that a low cost structure is designed and by reducing financial risk to the fabricator. Overall, the development of estimation software ensures the health of the steel construction industry and promotes steel as a structural material.

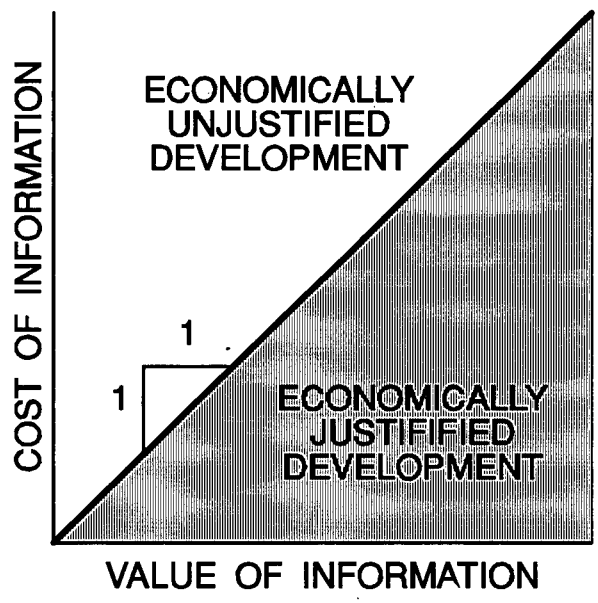


Figure 2: Economic Justification Of Development

CHAPTER 2

BASIC CONCEPTS AND METHODS OF COST ESTIMATION

A. Importance Of Cost Estimation

Cost estimation plays an important role in the design and fabrication of steel structures because the industry is economically driven. The owner, engineer and fabricator are all seeking the lowest cost alternate; however, each of the parties is motivated differently [fig. 3]. Cost is usually dictated by the fabricator who sets a price for the fabrication of a structure based on his prediction of cost and desired profit. The fabricator must be able to predict costs accurately and assess production strategies to prepare competitive bid prices. The engineer requires cost information to select between conceptual designs and refine details. In addition, the engineer must usually supply the owner with cost information so that the owner can evaluate the project and arrange financing. All three parties use cost estimates as a basis for decision making and require accurate information in order to reduce financial risk. Accurate estimation results in cost effective designs that improve the competitiveness of the steel construction industry. Inaccurate cost estimation can result in business losses, law suits, bankruptcy, and damaged reputations.

In order to realize the importance of cost estimation in the steel design and fabrication industries, it is necessary to appreciate the unique nature of steel fabrication in comparison with other types of construction and manufacturing. While most types of construction take place under field conditions, steel fabrication occurs under reasonably controlled shop conditions; however, these conditions can not be controlled

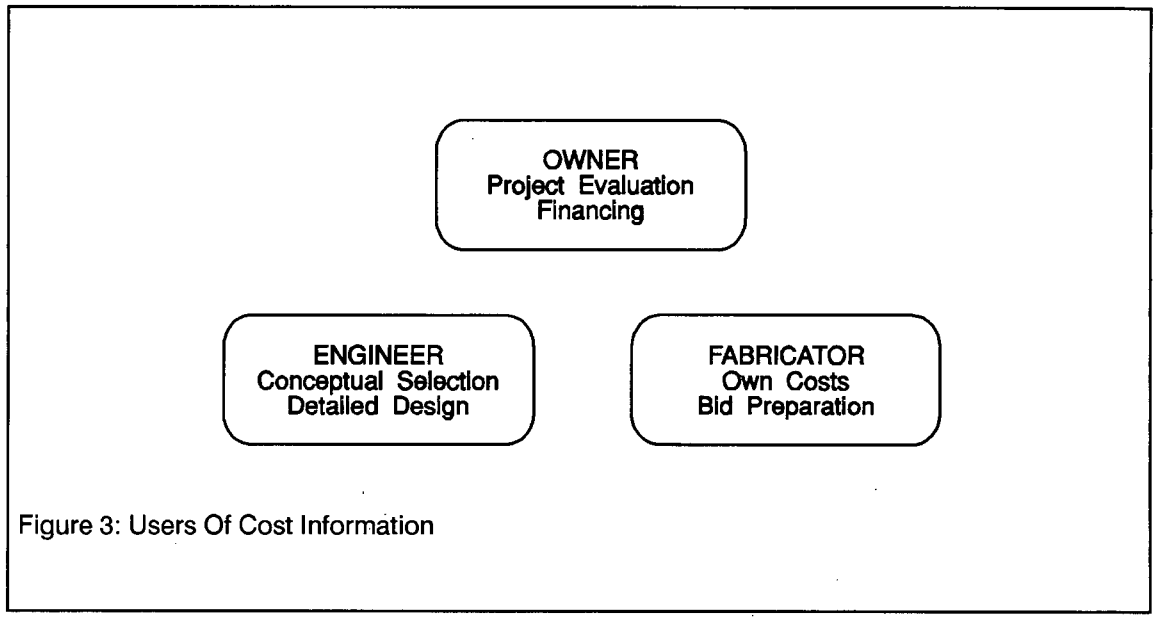


Figure 3: Users Of Cost Information

as rigorously as those found in conventional mass manufacturing industries. Manufacturing industries produce large quantities of identical items under carefully controlled factory conditions. They are able to produce prototypes and utilize economies of scale to optimize production and reduce costs. The fabrication industry produces fewer items and has no opportunity to optimize the fabrication process once production has started. The cost of prototypes is generally not justified and economies of scale are too small to be exploited effectively.

The importance of accurate estimation is amplified by the politics and strategy of the construction industries. Competitive bidding forces low margins for profit and contingencies which can be quickly consumed by inaccurate estimation. Fabricators sometimes bid low deliberately in an attempt keep labour and equipment busy. They may rely on extras or the substitution of lower cost alternates to turn a profit. The successful bidder is often the most desperate or least conservative fabricator who is willing to operate with very low margins. In these cases, knowledge of the probable

accuracy of the estimate is essential to prevent financial catastrophe. Cynical observations have been made that the contract tendering process is a card game in which the lowest hand wins and the winner loses. Sometimes, the low bidder is ignorant of the true scope of the work or its complications. Othertimes, the bid may be incomplete. Complete accurate well-informed estimates upon which to base contract bids are necessary to ensure the health of the steel fabrication industry.

Despite the importance of accurate estimation, little research has been performed in this area. Estimates are often compared with final production costs, but little active effort is committed to improving the accuracy of estimation. When cost is overestimated, the savings are accepted without thanks; however, underestimation can have serious financial consequences. In fact, the importance of accurate cost estimation is best appreciated by those affected by underestimation: owners with insufficient financing; engineers found liable for low estimates; and, fabricators forced to operate at a loss or into bankruptcy.

B. Difficulties Of Accurate Cost Estimation

Accurate cost estimation is surprisingly difficult to achieve because of five factors:

a) the availability of estimating expertise;

b) the limited resources available for cost estimation;

c) the variability of market conditions;

d) the difficulty in modelling the accrual of cost; and,

e) the difficulty in verifying the accuracy of an estimate.

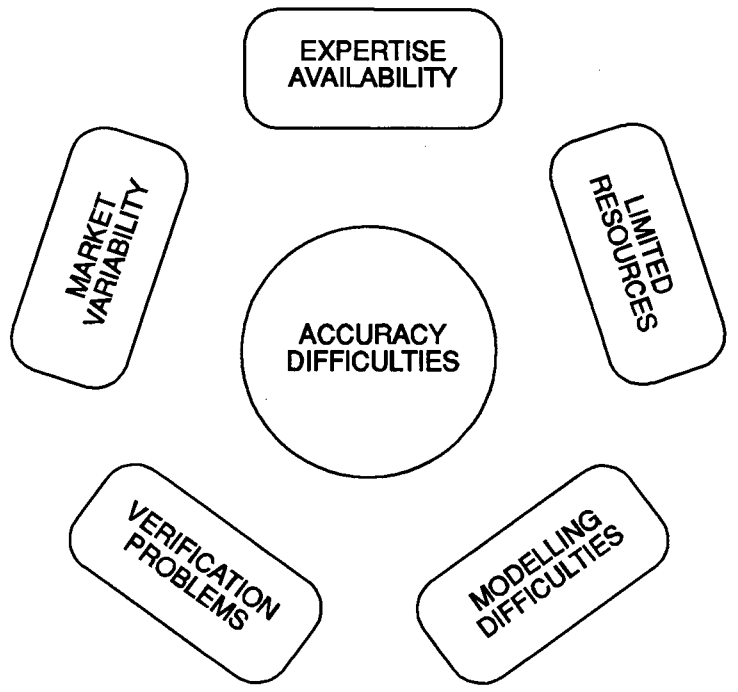


Figure 4: Cost Estimation Difficulties

Estimating expertise is often separate from design expertise. While most engineers possess a rudimentary knowledge of cost estimation, detailed estimation is usually performed by professional estimators because most engineers lack the specialized knowledge of fabrication techniques required to assess cost accurately. Unfortunately, professional estimators are usually assigned to special estimation sections which are

independent from the design sections of engineering and fabrication offices. This causes delays in communication between engineers and estimators which discourages interaction between the two groups because design decisions can not always be delayed until the cost information is available.

The economic constraints of a market economy limit the resources that are available for cost estimation. Each cost estimate requires a certain effort to produce, and this effort has an economic value. As the amount of effort committed to estimation increases, the quality of the information increases; however, at some point, the cost of acquisition exceeds the value of the cost information [fig. 5]. It is important to ensure that appropriate methods of cost estimation are used and that the effort committed to these methods provides a useful level of accuracy at a reasonable cost. Cost estimation is an expensive waste of effort if the results are too inaccurate for the comparison of alternates or if an unnecessarily high level of accuracy is sought.

The effect of market conditions on cost is, perhaps, the most difficult to understand and predict because it results from a complicated interaction between economics and psychology. The foremost difficulty is determining the difference between price and cost. The price is set by the fabricator and represents the cost to the owner; however, the price demanded by the fabricator may be quite different from his costs. The price is simply the value demanded by the fabricator for his services. It is governed by what the fabricator believes the owner is willing to pay and what the fabricator believes the prices of his competitors to be. In fact, the price set for a specific project may be a direct reflection of the cost of using an alternate construction material. For example, the cost of steel bridge girders may be set considering the cost of glulam beams and

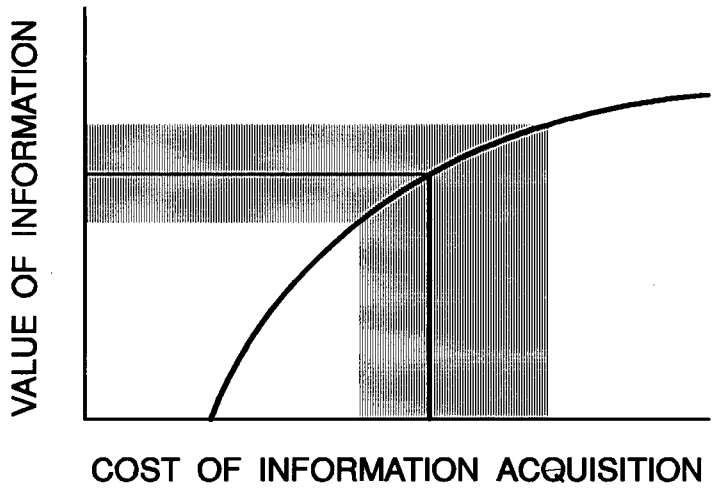


Figure 5: The Value Of Information Versus Its Cost

concrete box girders. There is no set relation between price and cost. Price may change rapidly while actual fabrication costs tend to be more stable. It is important for the engineer to predict costs to ensure that a reasonable low cost design is produced. If it is not, the fabricator may propose a lower cost alternate, but it is unlikely that he will pass along much of the savings [fig. 6]. The only way to ensure a low price is to design the lowest cost structure. In a competitive market, this will force the fabricator to bid near his actual costs because there will be no alternate proposals to reduce his costs after bidding. Unfortunately, in a monopolistic market, the fabricator may charge any price he can extort from the owner. On the other hand, in an overly competitive market, price may even be set below cost in order to keep labour and equipment employed. As a general rule, however, price is set slightly above cost in a healthy competitive market.

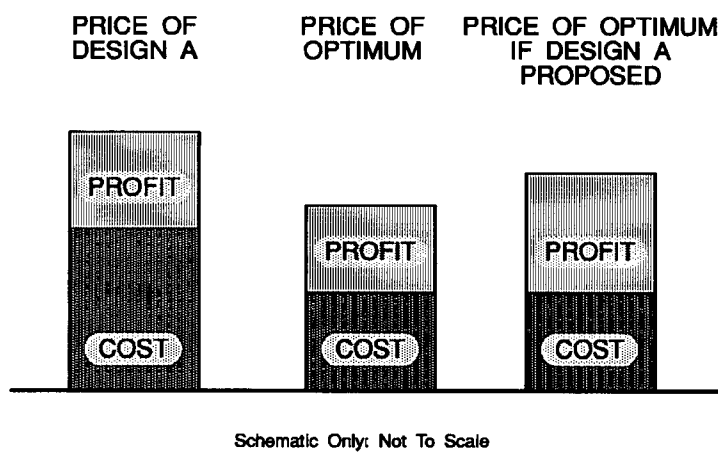


Figure 6: Cost And Profit Of Alternate Proposals

The modelling of cost is a crucial element of cost estimation and the choice of model is one of the major difficulties of the cost estimation problem. Every model is an approximation of reality [fig. 7]. It must be developed from an appropriate set of assumptions; otherwise, the predictions of the model will be in error. The assumptions of a cost estimation model must be based on some costing rationale which reflects the accrual of costs. While some costs can be attributed directly to a project, other costs are less tangible and must be allocated to a project on the basis of arbitrary policies. As a result, the cost of a project is dependent on the model used.

It is important to verify the accuracy of cost estimation because a minimum level of accuracy is required to ensure that estimates may be used as a reliable basis for selecting between design alternates. No one expects a cost estimate to be exact because it is only an educated guess at a random process; however, a reasonable level of accuracy

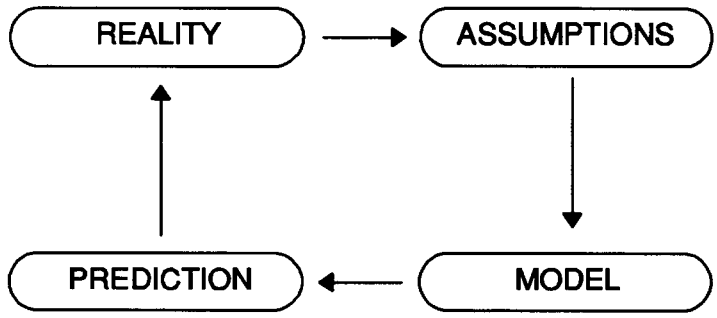


Figure 7: The Modelling Process

is expected. If this level is not achieved, the effort spent on estimation is wasted. Accuracy can only be verified using a statistical approach because a particular structure is only constructed once, and it is impractical to "repeat the experiment" a number of times to gain historical cost data and determine accuracy directly. This aspect of cost estimation has been largely neglected.

C. Modelling The Cost Estimation Problem

Modelling is the art upon which much of science and engineering is based. It is the process of reducing a real phenomena into a mathematical relation based on a finite number of parameters. A suitable model is needed as an abstraction to simulate the fabrication process in order to predict cost. The development of this model requires assumptions or axioms that may be justified or arbitrary. The first and most important

axiom of cost estimation is that cost is a quantity which can predicted with reasonable accuracy by a systematic evaluation of the project description. Although this assumption is normally accepted without question, it should be realized that the cost a project can be influenced by external factors. In addition, every project has some subjective qualities that defy quantitative description such as the type of assembly, the complexity of fabrication and the level of skill required.

The main problems in developing a cost estimation model are to establish a set of parameters to describe the project and to formulate a mathematical model to process them. In general, the mathematical model should be as simple as possible without compromising accuracy. The selection of parameters requires a good understanding of basic cost estimation procedures and an appreciation for the components of cost because it must contain all parameters with a significant influence on cost. The goal is to select a set which describes the project as completely as possible. An incomplete set of parameters will result in erroneous results because the model will be based on false premises.

i. Procedures Of Cost Estimation

No matter what mathematical model is used to estimate cost, the same basic procedures are used. Each model has three components:

a) a set of parameters to describe the project;

b) a source of background information including cost data; and,

c) an algorithm for compiling an estimate from the above information.

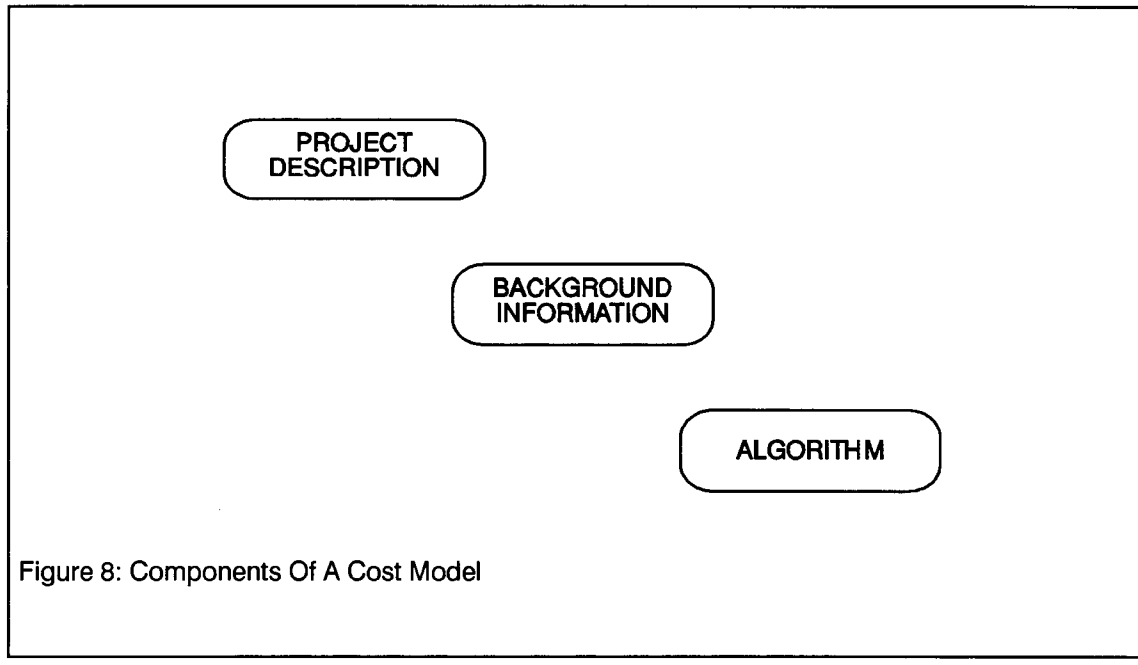


Figure 8: Components Of A Cost Model

Before any problem can be solved, it must be described; therefore, a project description must be prepared before a cost estimate can be compiled. Each description is a set of parameters describing various attributes of the project which influence cost. The more detailed the parameter set, the more accuracy can be expected. At the simplest level, a single parameter describing the size or scale of the project may be used, but this approach can lead to very crude results. A more complex approach is simulate the fabrication process in detail so that each item of material and operation of the fabrication process is accounted for directly. The parameter set is the sole link between the real project and the computer model or human estimator.

Background information is an important commodity in any expert task. Every human expert relies on background information in the form of personal experience and published information in order to assess and analyse problems. Similarly, a computer-based method requires a database of background information. Simpler methods of estimation may require only unit cost information, while more complex methods may require additional data such as a detailed knowledge of fabrication operations, productivity, and the availability of labour and equipment. Background information may be derived from accepted rules of thumb or from the intuition of an experienced estimator. Alternately, it may be compiled by detailed numerical analysis of fabrication shop records.

The final component, the algorithm, is the most essential and most overlooked element of cost estimation. Traditional methods all rely on a simple algorithm in which quantities from the problem description are multiplied by unit costs from the background information. This procedure is simple and usually supplies adequate information; however, it does not provide any information regarding accuracy. It was developed when manual arithmetic was the mainstay of cost estimation. The acceptance of computers by industry has provided an opportunity to introduce more complicated algorithms because it allows nonlinear equations which can be used to compute cost more accurately and stochastic methods which can be used to assess the interaction of operations and the variability of cost without a significant increase in direct human effort.

ii. Components Of Cost

The cost of fabricating a structure can be divided into two categories:

a) direct costs; and,

b) indirect costs.

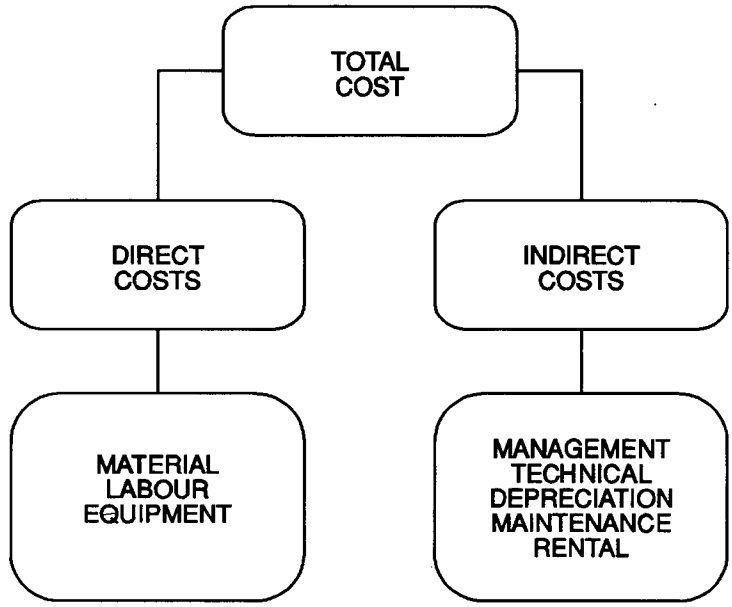


Figure 9: Components Of Cost

The division of costs into these two categories is an arbitrary matter of definition [fig. 9]; however, it is useful when considering how costs are apportioned. Direct costs are those costs which can be attributed directly to a specific item being fabricated while indirect costs are those costs arising from the general operation and maintenance of

the fabrication shop. The materials and operations used to prepare, shape and join contribute to the direct costs. Depreciation, rental, maintenance, technical and support costs are examples of indirect costs. These costs are real, but the rationale for apportioning them to various projects is less tangible and is usually determined by company policies and strategies.

Note that the price asked by the fabricator also includes profit and taxes. These can be a significant portion of the price but have not been considered in this thesis. Profit is difficult to estimate because it is highly variable and dependent upon market conditions. Taxes are usually a function of price or profit.

iii. Direct Costs

Direct costs are relatively easy to model once the level of detail to be considered is determined. Each direct cost can be associated with one or more physical parameters describing the material or fabrication operation required [fig. 10]. The quantity of each parameter can be determined in a quantity survey and a unit cost can be found for each parameter from historical data. The main short-coming of this approach is that it does not include the interaction between projects and activities on the shop floor. It assumes all activities are performed serially and that activities do not interfere with one and other.

DIRECT COSTS = f(PHYSICAL PARAMETERS)

Figure 10: Direct Cost Parameters

The levels of detail which can be considered are:

a) scale;

b) material; and,

c) material and operations.

At the simplest level of detail, scale, only the magnitude of the project is included as a project specific parameter. In a broad sense, the magnitude indicates the size and complexity of the project. This level of detail can not be extremely accurate, but it does give a ballpark cost estimate which may be used to determine feasibility and justify further investigation. The main benefit of this level is that it requires very little effort.

At the material level, the quantities of material required for fabrication are used as estimation parameters. These parameters give a more detailed description of the project and provide a more accurate cost estimate than the level of scale. This level of detail implicitly assumes that some typical quantity of work will be performed on each quantity of material.

The level of detail including material and operations is an attempt to create a model which closely simulates the fabrication process. The project is described in great detail

as one or more parameters are needed to describe each operation; hence, this level of detail promises the greatest accuracy. The cost of each operation is the combined cost of the material, labour and equipment required for that operation. The cost of labour and equipment is not usually separated because these components of cost are incurred simultaneously and are inter-related.

iv. Indirect Costs

Indirect costs are the most difficult to include because their apportionment requires the establishment of some policy regarding the distribution of each element of indirect cost. The manner in which each type of indirect cost is accrued often dictates the apportionment policy. Depreciation occurs constantly; however, the fabricator may choose to include depreciation over a long period of time or on a single contract depending on market conditions and the specialty of the equipment. Long term rentals on major equipment and space must be paid on a regular basis whether or not the facilities are used, so this type of cost may be included on every job. Short term rentals for particular projects can be included in the budget for that project. Maintenance is periodic and repairs must be done at random, so some policy must be designed to smooth out these costs and provide funds so that unexpected repairs do not consume profit. A contingency fund with contributions from each project can be established to cover this type of expense. Technical support can usually be attributed to specific projects even though it can not be attributed to individual items. General support such as office staff, supplies and janitorial services can not be attributed to specific projects

and must be paid for out of general revenue.

The fabricator may apportion indirect costs equally to each project or prorate them according to project size. One common method is to prorate the indirect costs on the basis of direct costs. In this method, the unit cost of each fabrication activity includes a direct and indirect component. No new parameters are introduced and the number of variables in the project description is minimized. Moreover, the parameters used may be determined directly from a quantity survey. This method assumes that larger and more complicated projects will be responsible for a larger share of the indirect services required to operate a fabrication shop.

Regardless of how costs are accrued, the fabricator may choose to subsidize the costs of certain projects with the revenue of other projects as part of an overall business strategy. Such a redistribution strategy may be intended to obtain contracts in difficult market conditions in order to minimize losses and retain skilled employees. On average, however, indirect costs must be recovered or the fabricator will suffer a loss.

D. Method, Effort And Accuracy

The primary objective of cost estimation is to predict the cost of a project with reasonable accuracy. The accuracy of a cost estimation method is a direct reflection of how effectively the parameters of the underlying mathematical model describe the components of cost. Models with a greater number of parameters tend to be more

accurate, but also tend to be more expensive because they require more effort. Unfortunately, the economic constraints which make cost estimation such an important part of the design and fabrication process also govern the estimation process. Thus, the optimum method of cost estimation is the one which provides the required accuracy with the least effort. As a result, there is an interaction between method, effort and accuracy which must be considered before any estimation is done [fig. 11].

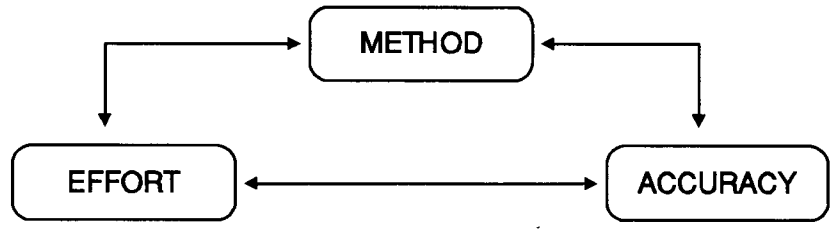


Figure 11: Method, Effort And Accuracy

A hypothetical graph of accuracy versus effort for a single estimation method [fig. 12] reveals many of the problems inherent in cost estimation. Firstly, there is a diminishing return for each additional unit of effort applied. Thus, while it is desirable to achieve high accuracy, the cost of achieving greater accuracy increases rapidly. At some point, the cost of obtaining more accurate cost information will exceed the benefit of obtaining it making it more advantageous to consider more alternates using a less precise method to ensure that a more optimum conceptual design is chosen for detailing; however, a minimum level of accuracy exists below which judicious comparisons may not be made. Secondly, there is an upper limit to the accuracy which

may be achieved using a particular method regardless of the effort applied. This means that a particular method may simply be unsuitable for accurate estimation because the so-called ultimate accuracy is too low. Thirdly, there is a minimum level of effort below which no results are obtained.

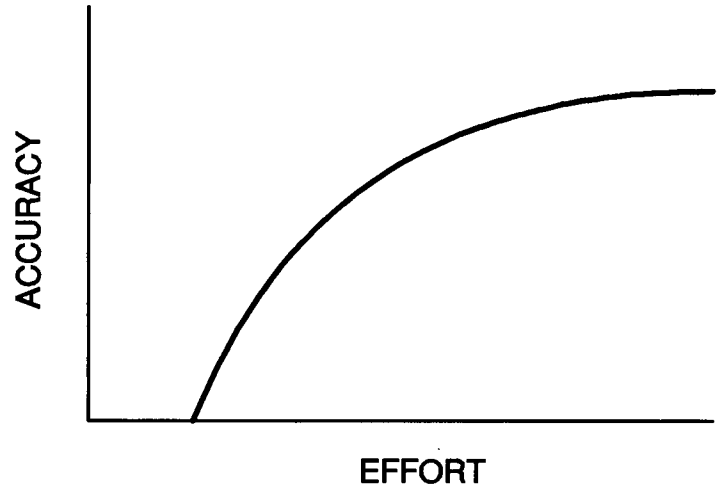


Figure 12: Accuracy Versus Effort For A Single Method

When the accuracy curves of available methods are superimposed [fig. 13], it is apparent that there is an optimum combination of method and effort for each desired level of accuracy. The optimum combination is the one which provides the desired accuracy with the least effort. Note that there is no correlation between the ultimate accuracy of the methods or the minimum effort required to produce an estimate and the optimum combination. While there is generally only one optimal combination,

there may be several methods with the capability of producing estimates. Conversely, some methods may be incapable of achieving the desired accuracy. Accuracy may also be limited by the availability of resources. When it is not possible to achieve the desired accuracy within the resource constraints, the method providing the greatest accuracy using the available resources must be accepted. Alternately, greater accuracy may be achieved by one of three methods:

a) additional effort may be applied to the problem;

b) technological aids may be introduced to magnify effort; or,

c) a new method may be developed.

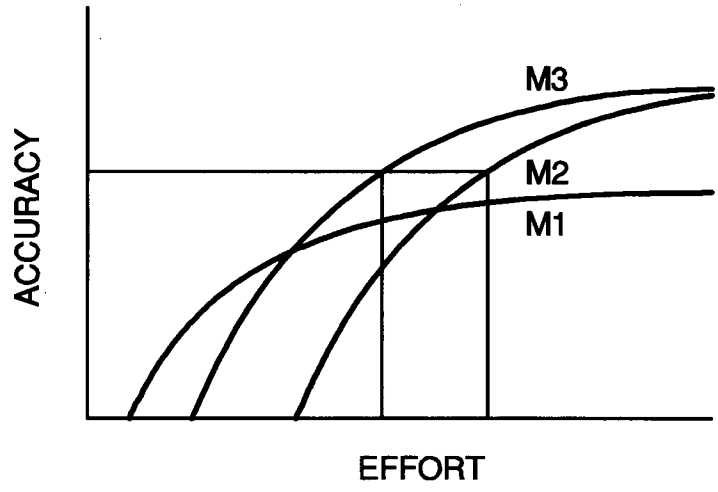


Figure 13: Accuracy Versus Effort Superimposed For Several Methods

If any of the above changes are made, the optimal combination of method, effort and accuracy is altered. Applying more effort is often an effective short term approach to the problem provided the accuracy versus effort curve is not too flat. Introducing technological aids improves the accuracy that can be obtained by the available resources, but it requires initial research and development. Developing a new method requires creative effort but may be essential if the desired accuracy is beyond the limits of current methods. Traditionally, additional effort is applied to the problem until it becomes apparent that the increased effort is not an effective way to increase accuracy. Then, economic pressures justify the investment necessary to develop tools to aid in cost estimation. New methods are only developed when the need for greater accuracy forces developments to be made.

E. Existing Methods Of Cost Estimation

Many different methods of cost estimation exist; each based on its own mathematical model and best suited to a specific purpose. It is important to select a method keeping in mind the expected economic return of the estimate. In general, the larger and more complex the project, the greater the expected return. Small simple projects may not even warrant estimation. The cost of estimation itself may exceed the benefit of selecting the most economic design when inappropriate methods of estimation are used.

Methods of cost estimation may be classified by considering the type of parameters used to describe the structure in the mathematical model. The three basic classes identified using this method of classification are:

a) estimation by characteristic parameter;

b) estimation by units; and,

c) estimation by operations.

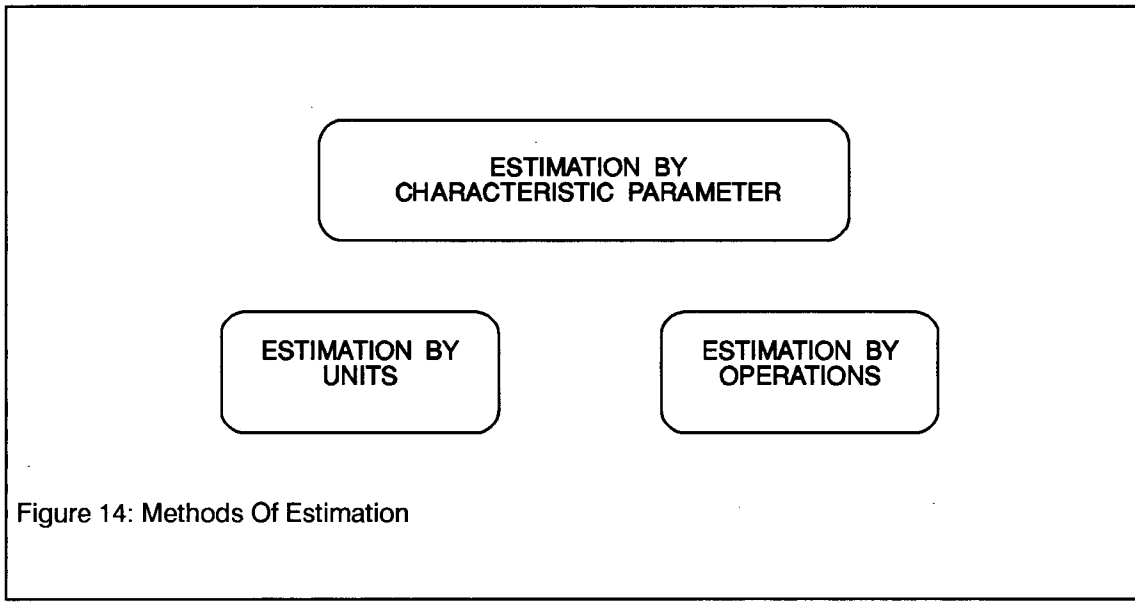


Figure 14: Methods Of Estimation

i. Estimation By Characteristic Parameter

Estimation by characteristic parameter is a very coarse method which is often used to assess costs during the conceptual design stage. It uses a very low level of detail that reflects only the scale of the project. Only one generalized characteristic parameter

and a generalized unit cost are used to predict the total cost. Very little effort is required because little project specific information is incorporated into the estimate. Moreover, it is not even necessary to have completed the design before estimation. The accuracy of this method is highly variable as it depends upon the subjective judgement of the estimator to produce a single unit cost which reflects the contributions of material, labour and equipment. Nevertheless, reasonable estimates may be obtained if skilled and experienced personnel are available.

Examples of this method are the cost estimation of a single storey steel warehouse on the basis of plan area or a bridge on the basis of steel tonnage. These structures can often be placed in classes based on the type of construction used making it easier to select unit costs that reflect the connections and details. Estimation by characteristic parameter may be quite accurate for well defined types of construction but is less reliable under other conditions.

The main virtue of estimation by characteristic parameter is its simplicity. It is easy and quick to use; however, this virtue is at the expense of accuracy. Estimation by characteristic parameter can only be recommended for the comparison of well-defined types of construction when experienced estimators are available. Under other conditions, it is too inaccurate for comparison but may provide rough estimates. It is not well suited for computer application because it relies of subjective judgement for accuracy.

ii. Estimation By Units

Estimation by units is more accurate because more project specific information is incorporated into the estimate and less judgement is required. This method uses parameters which reflect the quantity of material in the structure. A separate unit cost can be used to represent different types of material. Additional effort is required because more project specific information is included and the preliminary design must be largely completed. It is more accurate than estimation by characteristic parameter but still relies on the experience of the estimator to include the effect of labour and equipment. Good results may be obtained when typical connections and details are used.

An example of this method is the cost estimation of a frame by subdividing the structure into beams and columns. Separate unit costs may be used for beam, column and bracing material to reflect the different sections and fabrication required. Cost data from previous projects and the judgement of the estimator can be used to select unit costs.

Estimation by units is a relatively accurate method of estimation. It requires much more effort than estimation by characteristic parameter but can be used to compare alternates of similar construction because the quantity of material is explicitly included. While this method is not entirely well suited for computer application because it requires a large degree of judgement for the selection of unit costs, computer technology can be used to transfer and manipulate the large quantities of data used.

iii. Estimation By Operations

Estimation by operations is the most accurate method because it explicitly models the fabrication process as well as the materials used. In this method, the structure is first divided into major components and then further subdivided into the materials and operations required to fabricate each item. A very large amount of project specific data is incorporated into the estimate; consequently, this method is the most accurate of the three methods. Unfortunately, it also requires the greatest effort to compile. Estimation by operations does not rely on subjective judgement. Accurate estimates may be compiled by any estimator who has access to productivity and cost information.

In the case of estimation by operations, the example frame structure is divided into individual members. The materials required for each member are then determined and a material cost is found using the current material prices. The operations required to fabricate each member are assessed and a fabrication cost is found using the appropriate unit costs for operations. Finally, the material and fabrication costs are combined to complete the estimate.

Estimation by operations is the most accurate method, but it also requires the most effort. In addition, it requires access to a large database of cost data which must be derived from available records and periodically updated in order to ensure accuracy. Unfortunately, the cost database is unique to each fabrication shop. This method may be used to assess atypical structures because such structures are generally fabricated using typical operations. It is particularly useful in determining the consequences of

altering connections and details because such changes may affect cost significantly without altering material requirements. Estimation by operations is best suited to computer application because it relies least on subjective judgement.

CHAPTER 3

BASIC CONCEPTS IN COST ESTIMATION AIDS

A. Definition Of Estimation Aids

An estimation aid is any technique or device used to increase the effectiveness of cost estimation. It is a tool which is used to focus, replace or magnify the effect of direct human effort. An aid may be as simple as printed forms or as complex as a fully integrated knowledge based expert system on a computer system.

B. Purpose Of Cost Estimation Aids

The purpose of a cost estimation aid is to improve some aspect of cost estimation. It alters the constraints of the accuracy versus effort diagram [fig. 15] resulting in a new optimum combination of method, effort and accuracy. Estimation aids may be used either to increase the accuracy that can be achieved using available resources or to reduce the effort required to achieved a desired level of accuracy.

The use of an estimation aid shifts the accuracy versus direct human effort graph to the left. This represents savings in direct effort and increases the effectiveness of applied effort. If the graph is shifted left by amount A, an increase in accuracy of amount B can be expected if the same amount of direct effort is applied. Alternately,

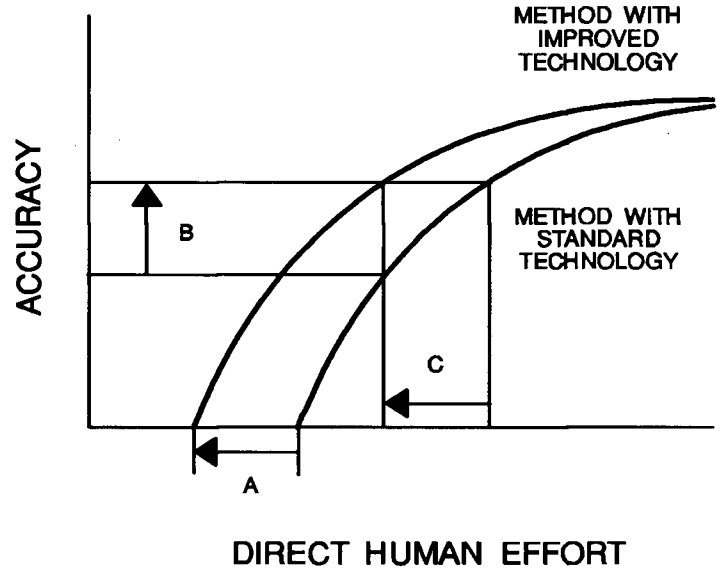


Figure 15: The Effect Of Technology On Effort Versus Accuracy

the same degree of accuracy can be maintained with a savings in effort of amount C which is equal to amount A. The use of an estimation aid does not alter the shape of the curve; it only alters the relative economy of available estimation methods.

It is important to remember that cost estimation is a tool used in the design and fabrication of steel structures. Although, the engineer and fabricator both commit significant resources to cost estimation, it does not contribute directly to the final product. The engineer rejects many concepts and details after cost estimation and the fabricator's bid is often rejected despite the time and effort committed. Therefore, it is desirable to minimize the cost of estimation so that more alternates can be considered by the engineer and the fabricator is exposed to less financial risk. Cost estimation

aids increase the efficiency of industry by reducing the cost of estimation and allowing more alternate proposals to be investigated. The economic benefits resulting from the use of estimation aids justify their use and development.

C. Evolution Of Cost Estimation Aids

Cost estimation is an expanding field of knowledge. It is also a complex field in which relatively primitive methods can be found working alongside state-of-the-art technology. The sophistication of the tools has increased drastically since the advent of the computer revolution; however, the dissemination of these tools has been limited because many have been developed as the proprietary technology of various design and fabrication companies.

The evolution of cost estimation aids may be divided into four phases which reflect the historical introduction of technology into the cost estimation process. In the order of increasing sophistication, these phases are:

a) the primitive phase;
b) the simple calculation device phase;
c) the data manipulation phase; and,
d) the expert system phase.

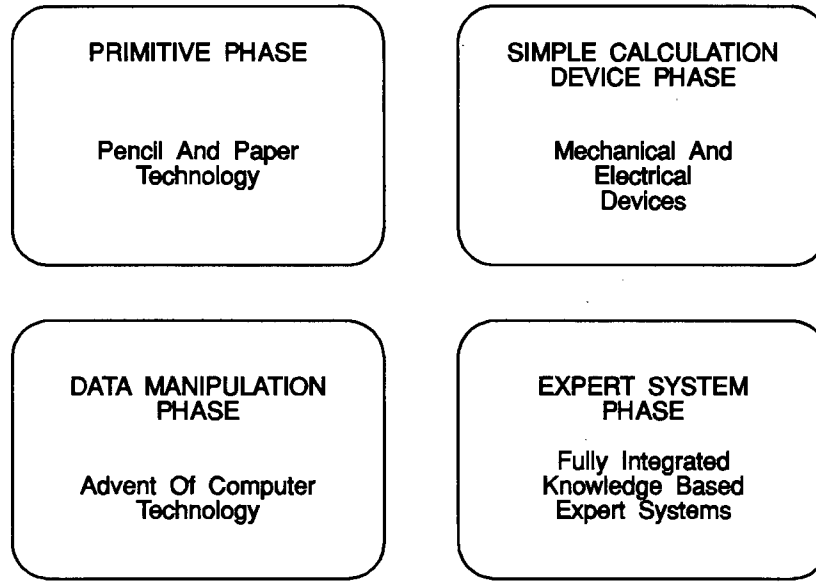


Figure 16: The Development Of Cost Estimation Aids

The primitive phase is the simplest and is best described as the "pencil and paper" phase. This phase essentially relies on mental power for all calculations and data manipulation; however, it includes several significant cost estimation tools:

a) paper forms that can be used as templates;
b) mathematical tables to increase the efficiency of calculation; and,
c) catalogues containing cost information.

This phase is best implemented using methods with a low level of detail because it becomes slow and uneconomic quite quickly when a large number of calculations are required. It can be used effectively with the estimation by characteristic parameter

method because this method relies on subjective judgement rather than detailed calculations to include the specialized attributes of the structure; hence, a large number of calculations are not required. This type of estimation may be appropriate when a very quick approximation of cost is required.

The simple calculation device phase represents a significant improvement over the primitive phase as all calculations are performed using mechanical or electronic tools. Calculators, adding machines, and sliderules have all been used to aid calculation. These devices increase the speed and reliability of calculation, but have no effect on other aspects of cost estimation. These devices make it feasible to use more complex cost estimation methods than can be attempted using primitive means because the effort required for calculation is greatly reduced.

The data manipulation phase incorporates all the advantages of the previous phases with the data manipulation capabilities of the computer. This represents a significant improvement over other methods because it couples the manipulation and calculation abilities of the computer with the judgement and skill of an experienced cost estimator. Most existing estimation programmes including spreadsheet and data base programmes are in this phase.

The expert systems phase is the present frontier of cost estimation. An expert system is a computer programme which attempts to emulate the decision making capabilities of a human expert. The development of such a programme requires an algorithm which allows the computer to make decisions which normally require subjective assessment. This is difficult to achieve due to the rigid logic system imprinted on the computer

circuits; however, the difficulties can be minimized by coupling the system to an estimation method which minimizes subjective decision making. Thus, the method of operations is best suited to such a system. The expert systems phase does not require an experienced operator as the entire estimation process takes place within the machine. This phase is the most efficient because all decisions, calculations and manipulations occur at high speed without interruptions for operator input.

D. Computer Capabilities And Cost Estimation

The computer is the most important, and versatile, tool for cost estimation. It is capable of replacing all previous estimation aids, including pencil and paper. With the use of expert systems techniques, computer technology can even replace the skills of the estimator himself. However, the full benefit of computer technology can not be realized until the capabilities of the computer environment are fully understood and the limitations imposed by manual methods are stripped from current cost estimation methods. Once this is achieved, computer-aided estimation will be far more efficient than existing methods.

The integration of cost estimation and computer technology is a natural continuation of the new industrial revolution brought about by the introduction of the computer into the office environment. Like the original industrial revolution, its purpose is to increase the productivity of each worker through the use of technology. The benefits of computer technology are achieved by exploiting the strengths and avoiding the

weaknesses that are inherent in the computer environment [fig. 17].

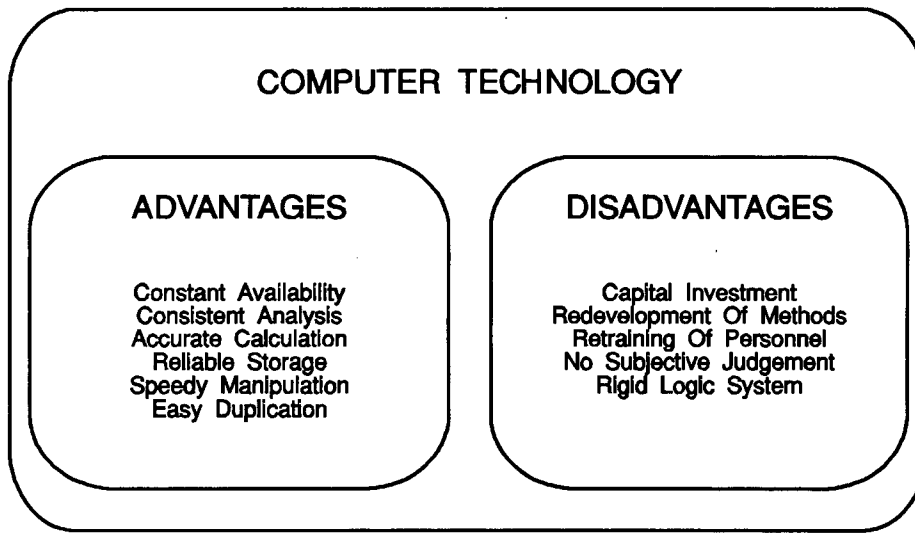


Figure 17: Advantages And Disadvantages Of Computer Technology

The principal strength of computer technology is its ability to manipulate large quantities of data quickly and accurately. The computer is able to store, access and transfer large volumes of data much faster and with less probability of error than a human estimator. It can also perform the mathematical operations necessary for cost estimation much more efficiently than the human mind. In addition, the rigid logic of computer technology ensures that its task is performed consistently and without bias. A human estimator can easily become bored by repetitious tasks or preoccupied by external matters leading to errors in data manipulation or calculation. For these reasons, computer generated estimates are more suited for comparison than manually

generated estimates.

The principal weakness of the computer is that its rigid logic system is incapable of abstract thought. This weakness can be avoided in three ways. First, cost estimation systems can be developed in which the computer performs all data manipulation and calculation operations while the estimator provides the necessary subjective assessments. Secondly, estimation methods can be developed which minimize the need for subjective assessment. Thirdly, expert systems can be developed that superimpose new logical systems on the computer's existing logic system and allow subjective assessments to be made directly by the computer. This requires an exhaustive analysis of the problem and the development of a suitable logical system.

Before the limitations imposed on current methods of cost estimation by manual methods can be removed, they must be identified. There are two basic types:

a) the availability of expertise; and,
b) imposed systematic limitations of thought.

In a general sense, the availability of cost estimation expertise depends on market conditions; however, availability is also affected by the previous commitments, the health and holidays of the cost estimator. The use of computer-based cost estimation systems improves the availability of expertise. In addition, some systems increase the effectiveness of cost estimation. Expert systems eliminate the need for direct consultation with experts. The use of such systems ensures that cost estimation expertise is available on an "on call" basis. In addition, computer-based systems are

easily duplicated and distributed. Unlike human estimators, they can be stored without a retaining fee and do not lose their effectiveness when left unused for long periods of time.

Systematic limitations of thought are a form of tunnel vision imposed on new cost estimation systems by constraints which are no longer relevant. Methods requiring detailed or complex calculations were impractical before the introduction of computers to the office environment. These methods are now practical because the use of new technology has caused a shift in the optimum combination of method, effort and accuracy. Unfortunately, many computer-based cost estimation systems are direct analogs of manual cost estimation methods. The development of computer-based cost estimation aids must be freed from these limitations of thought so that efficient computer-based systems can be developed.

The realization of the benefits of computer technology will result in fast, accurate and convenient cost estimation systems that allow mental effort to be redirected to the more creative and subjective aspects of design and fabrication such as the optimization of structural systems and production methods. Reduced cost and increased availability will increase the significance of accurate cost estimation. The improved quality of information will result in better economic decisions resulting in a healthier steel industry. The shear economic effectiveness of computer-based estimation methods will encourage the acceptance of these methods by industry.

CHAPTER 4

ADVANCED CONCEPTS FOR COST ESTIMATION: STOCHASTIC; SENSITIVITY; AND, SIMULATION METHODS

A. Advanced Concepts For Cost Estimation

The introduction of computer technology to the field of cost estimation provides an opportunity to develop more advanced methods of cost estimation. Computer technology provides the power to develop methods of cost estimation which can not be performed practically using manual methods. Three areas in which computer technology might be integrated with considerable benefits have been identified for detailed consideration:

a) stochastic methods;
b) sensitivity methods; and,
c) simulation methods.

The incorporation of these methods into computer-based estimation tools will provide information that is normally inaccessible using conventional cost estimation techniques. These concepts will enable the probable accuracy of an estimate to be calculated statistically, the key elements of cost to be isolated, and the fabrication process to be optimized. At present, no established analysis techniques are used in these areas and experts must rely on their experience, intuition and judgement. In the future, estimators may consider such information to be indispensable.

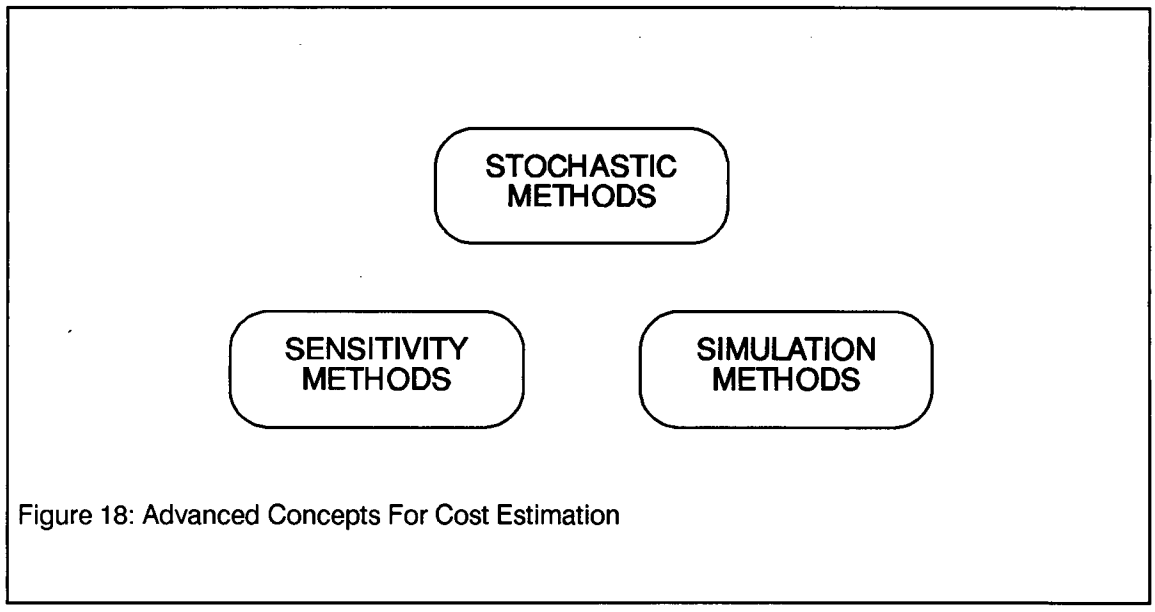


Figure 18: Advanced Concepts For Cost Estimation

B. Stochastic Methods

Stochastic methods recognize the uncertainty inherent in any estimation procedure. They are based on the same probabilistic models that are used for reliability-based design in other areas of civil engineering. Stochastic methods recognize that the variables used in the cost model are randomly distributed and that it is unlikely the actual cost will be exactly equal to the estimate. Monte Carlo simulation or analytical statistical methods can be used to determine the overall distribution of cost rather than a single data point. This provides important additional information which can be used to compare alternates and prepare bids.

The traditional estimation methods used in the steel design and fabrication industries implicitly evaluate mean costs. Mean costs are most meaningful when dealing with large production volumes as in the manufacturing industries. They are not suitable in the fabrication industry where a single structure is produced. Variability is extremely

important because of the actual cost of an item can be significantly different than the mean prediction when small volumes are produced. Stochastic methods provide information which can be used to establish confidence limits defining the likely range of variation about the mean estimated value. The importance of such information is critical to the construction industries because contracts are often awarded on a lump sum basis. Stochastic methods minimize financial risk because they describe variability explicitly.

Stochastic methods can also be used to check the effectiveness of cost estimation techniques. When traditional methods are used, an estimate is prepared and the structure is built, but there is no formal method of verifying the success of the estimation method. Results are accepted on faith and no attempt at verification is made despite the significance of cost information on major financial decisions. Stochastic methods can be used to evaluate the effectiveness of cost estimation methods and justify comparison and optimization of alternate designs. Moreover, verification will allow the weaknesses of various cost estimation methods to be identified and eliminated.

Cost estimates are often used to compare alternate concepts or details. Traditionally, only mean costs have been compared; however, direct comparison of the mean costs can not be justified unless the estimates have similar coefficients of variation. Knowledge of the variation of costs is as important as knowledge of the mean. Even when estimates have the same coefficient of variation, the likely variation about the mean may be large enough that no significant statistical basis exists for a decision. Estimates with dissimilar coefficients of variation can only be compared while

considering the likely variation of costs; otherwise, decisions may be made on the basis of incorrect evidence. Stochastic methods provide the information necessary to make judicious comparisons.

The importance of this information is illustrated in the following figure [fig. 19]. In the first graph, there are two estimates A and B with similar mean values but with dissimilar coefficients of variation. The distribution with the larger coefficient of variation B has a much larger range of likely costs. It may be advisable to select the alternate with the smaller coefficient of variation A as it has a smaller likelihood of exceeding a given value. In the second graph, the distribution with a the lower mean cost D has a much larger coefficient of variation than the other distribution C. A decision based solely on mean values would undoubtedly prefer alternate D; however, a decision based on both mean cost and variability requires a more complex set of criteria. Alternate C may well be preferable if it has a lower chance of exceeding a given cost value. In other words, it may be worth paying a premium in terms of expected mean cost in order to reduce the probability of paying a higher actual cost.

C. Sensitivity Methods

Sensitivity methods are numerical methods which are used to increase the efficiency of optimization techniques. These methods determine the contribution of each input variable to total cost so that the available resources can be allocated most effectively to the refinement of the structural design and fabrication process, thereby, minimizing

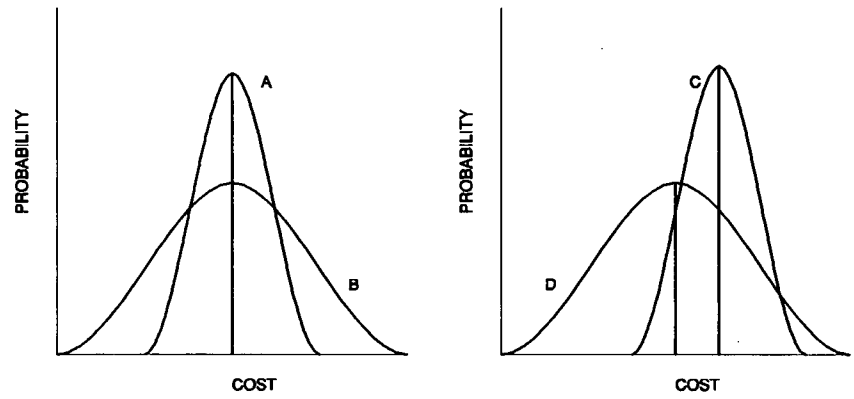
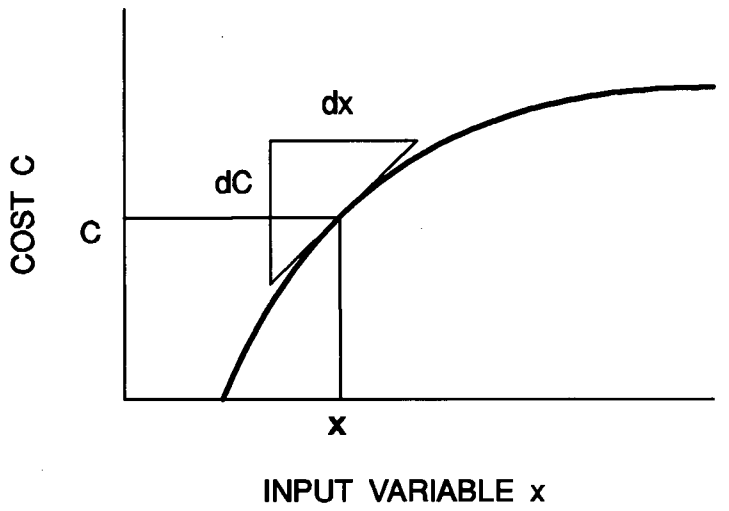


Figure 19: Comparison Of Cost Distributions

overall cost. In the design stage, the engineer can use sensitivity analysis to identify critical structural elements so that less expensive alternates can be investigated. The fabricator can use the analysis to evaluate his production methods in order to identify critical operations. The significance of these operations can then be altered by adding new equipment to increase production capacity, rearranging the sequence of operations into a more efficient configuration or selecting an alternate component or production strategy. Sensitivity methods are used to maximize the total savings possible using the limited resources available. It provides a prediction of savings which can be used to ensure that the cost of further investigation does not exceed the likely savings.

The sensitivity of a particular input variable is measured as a normalized marginal cost [fig. 20]. Mathematically, marginal cost is the tangent slope or derivative of the cost function with respect to the variable being considered. It represents the increase in total cost which can be expected by an additional unit of the input variable. Note,

however, that it is a linearization of the cost function and is only a valid approximation for small changes in the input variable. Marginal cost is normalized by dividing by total cost to calculate sensitivity. This produces a value which is scaled with respect to the importance of each input variable.



$$\text{NORMALIZED MARGINAL COST} = \frac{dC/dx}{C}$$

Figure 20: Marginal Cost

In most structures, it is the connections and details rather than the materials which contribute most significantly to total cost. While traditional design methods rely on intuition and experience to select cost effective details and connections, sensitivity methods can be used to analyse costs directly. These methods assume that the greatest

savings will be achieved by reducing the cost of those items which contribute most significantly to total cost. Obviously, the savings produced by the refinement of any detail can not exceed the total cost of that detail. Therefore, it is best to concentrate available resources on the refinement and redesign of the most significant items. Furthermore, sensitivity analysis can be used to determine the feasibility of further refinement. At some point, it becomes more beneficial to accept an overly conservative design rather than undertake the expense of refinement.

D. Simulation Methods

Simulation methods are an attempt to model the fabrication process directly. These methods are usually suitable for use only by the fabricator because the elements of such a model require an intimate knowledge of available equipment and its operating characteristics. Most likely, a proprietary model is required for each fabrication shop. This type of model is usually highly non-linear as it reflects the interaction of operations in the fabrication process. It is the preferred type of model for optimization of production.

Simulation methods create models of the labour and equipment operation within a computer environment. Each machine or unit of labour represents a production resource which can be arranged sequentially, in parallel or concurrently to model various production strategies. Rearrangement of production resources within the programme allows several production strategies to be assessed without the

commitment of actual labour and equipment: an opportunity that was unavailable before the development of computer-based simulation. Simulation reduces production cost by ensuring that equipment is utilized as efficiently as possible.

Fabrication processes may occur sequentially, in parallel, or concurrently. The interaction of processes seldom allows the full capacity of each operation to be fully exploited. Some fabrication operations act as a bottleneck and reduce the overall efficiency of the other operations by causing them to sit idle between operations. This is an unavoidable waste which increases costs without contributing to production; however, it can be minimized by careful sequencing and planning which minimizes the interference of fabrication operations with one and other, thereby increasing productivity and profit. Optimization of sequencing must consider the effect of increasing capacity, altering production strategies and stockpiling of components to even the flow of production.

Traditional methods of sequencing fabrication rely on the experience of shop personnel. These methods must assume production rates based on the capacities of the equipment used and are not usually capable of directly assessing the interaction of equipment in order to calculate specific production rates. Thus, traditional methods do not fully utilize equipment. These methods may produce reasonable results, but better results can be obtained using true simulation methods.

CHAPTER 5

AN ADVANCED TOOL FOR COMPUTER-AIDED COST ESTIMATION: THE EXPERT SYSTEM

A. Definition And Purpose

An expert system is a computer programme designed to emulate the experienced-based decisions and recommendations of a human expert. It is a tool used to process concepts that are more abstract and logical systems that are less rigid than those found in conventional computer programmes. Expert systems store captured expert knowledge in a computer format so that it can be preserved, applied and manipulated without the need to consult human experts. They can be used:

a) to retain and preserve the knowledge of human experts;

b) to apply the knowledge of experts when they are unavailable in person;

c) to increase the availability of expert knowledge when it is in great demand;

d) to disseminate expert knowledge quickly and inexpensively;

e) to reduce the time required for expert assessment of a problem;

f) to provide a means of training new experts; and,

g) to aid established experts in their duties.

B. History

Expert systems were originally developed in the laboratory as tools for artificial intelligence research. These tools were first applied to simple problems in well defined environments. It was not until the 1970s that expert systems were developed for problems in the real world; however, expert systems are now available commercially for tasks in many fields including medicine, geology, engineering, commerce and law.

MYCIN, probably the best known system, was one of the first available expert systems. It was developed at Stanford to aid physicians in the diagnosis of bacterial diseases by determining the possible causes of a set of symptoms and recommending appropriate tests to differentiate between the possible causes of illness so that the actual cause could be identified. MYCIN demonstrated the feasibility of using expert systems for real problems. Later, its database was removed and the remainder of the system was marketed as the expert system shell EMYCIN. This shell has been used in the development of a number of other medical and non-medical expert systems.

Expert systems have also been developed for many tasks in civil engineering. HYDRO was developed for interpreting hydrologic data and CONE was developed for evaluating soil stratigraphy and strength properties using cone penetration data. However, expert systems seem to have generated the most interest in structural engineering. Systems such as DESTINY and HIRISE assist in the planning and design of structures while SACON provides advice for finite element analysis. Another system, SPERIL, evaluates siesmic damage to structures. While many programmes have been

developed to aid cost estimation, an expert system for cost estimation does not appear to have been developed, although it is possible that such a tool has been developed as a proprietary tool by a steel design or fabrication company.

C. Advantages Of Expert Systems

The expert system approach has many advantages over traditional methods: human experts and conventional programmes. The concept of the expert system encourages a general understanding of problem solving strategies and a provides insight into a particular problem. The acquisition of expert knowledge in a computer format enables information to be preserved and applied indefinitely. Expert advice is always available without delay because, unlike human experts, the computer-based system has no outside commitments, vacations or other assignments. Perhaps the most significant advantage of an expert system is that it has no personality. It is incapable of fatigue, boredom or personal bias. An expert system executes its task consistently making it an ideal foundation for the evaluation of alternate scenarios.

Expert systems utilize all the features of conventional computer-based systems. They are able to take full advantage of the speed and reliability of computer hardware to perform accurate calculation and data manipulation. Their speed and capacity can be easily upgraded as more powerful hardware becomes available. In addition, they can be stored without maintenance and transmitted electronically as required. Moreover, their expertise can be duplicated quickly and easily to meet demand unlike human

experts who may require years of training.

D. Anatomy Of An Expert System

An expert system has three basic components to emulate the decision making capabilities of a human expert:

a) an inference engine;
b) a knowledge base; and,
c) a user interface.

These components ref lect key elements of a human expert. The inference engine replaces intellect while the knowledge base replaces memory. The user interface performs the all important task of communication which is vitally important to both human and computer-based experts.

i. Inference Engine

The inference engine is the key component of any expert system. It is the decision-making element containing the logic system used to manipulate the knowledge base and control programme flow. A successfully developed inference

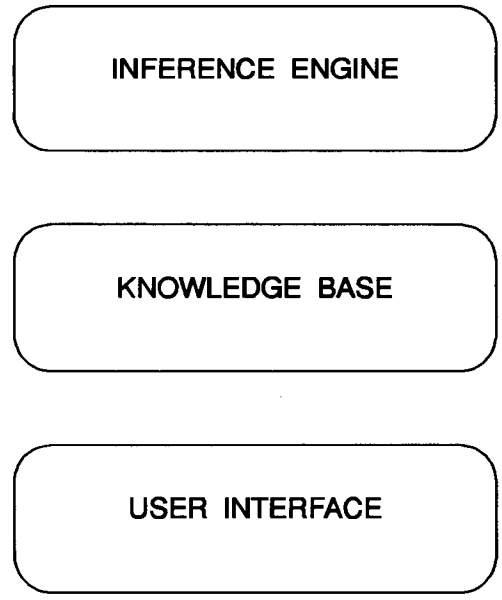


Figure 21: Components Of An Expert System

engine can be used in the creation of several expert systems because its general problem solving techniques are applicable to many fields of knowledge. This unique trait differentiates expert systems from conventional programmes which are inflexible because they are intended for a specific problem.

Many different logic systems can be incorporated into an inference engine; each system is best suited to a different approach to expert system programming. The most common system, binary boolean logic, is the basis of conventional computing systems. It is well suited to rigid tasks with little uncertainty; however, most expert problems involve some degree of uncertainty. One method of incorporating uncertainty is to use a logic system with three states: true; false; and, mu. Mu represents an indifferent or uncertain

state which is used when there is insufficient information to answer a question or when a question is meaningless. Another system, fuzzy logic, uses values between zero and one to represent the certainty of each state. Unlike conventional logical systems, fuzzy logic can be used when the facts are not yet certain, but information exists to indicate the probability of various facts.

While inference engines can be extremely efficient at isolating solutions to problems using established techniques, they are not yet capable of deriving new and creative solutions; therefore, every inference engine requires a well defined problem solving strategy. This is usually based on a strategy used by human experts. The isolation of this strategy is a topic in itself within the expert systems field.

There are three recognized classes of problem solving strategies: forward chaining; backward chaining; and, rule value methods. Forward chaining is a problem solving strategy in which the solution is constructed from the available data. Backward chaining is a strategy in which all known solutions are considered in light of the available information. The rule value method is a more complex method which combines the techniques of forward and backward chaining in an attempt create an optimum problem solving strategy. Rule value techniques attempt to determine which information is most critical to the solution of the problem so that the most efficient path can be taken towards solution.

ii. Knowledge Base

The knowledge base is the only source of data available to the expert system. It must contain all the information the inference engine requires to solve a particular problem. Every knowledge base must contain three separate sets of information:

a) a problem description;

b) a database of background information; and,

c) a set of rules governing the problem.

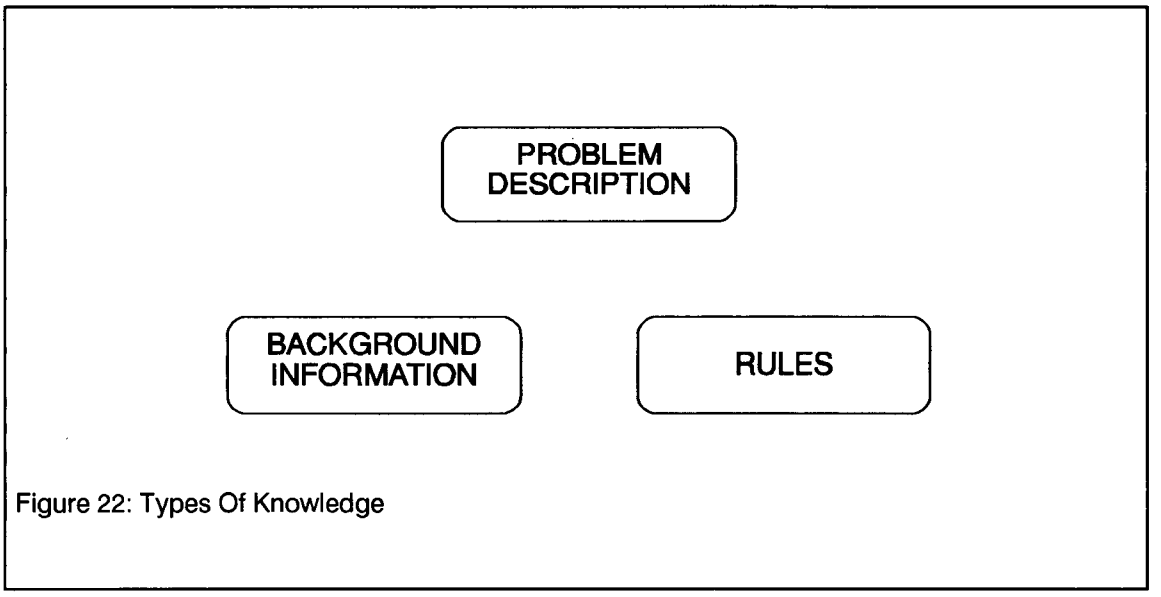


Figure 22: Types Of Knowledge

A description of the problem is obviously required before any solution can be attempted regardless of whether a problem is analysed by a human or computer-based expert. The description must be made in terms that can be interpreted as parameters for use in the model. Often, the key to an expert's success is his ability to translate the problem into model parameters. He must know what questions to ask in order to

identify and describe the problem. The process of reducing a physical problem into an appropriate set of parameters requires expert knowledge in itself, so part of the expert system must be committed to the interrogation of the end user in order to identify and describe the problem. A well designed user interface can help streamline this task.

Background knowledge is the key commodity of a human expert. Generally, experts are no more intelligent than other human beings, they are just more familiar with the problem being considered. Experts know where to find the appropriate literature and have access to a large store of personal knowledge and experience. The database of background information supplies the expert system with general information that is required to solve a problem but is not part of the problem description or part of the solution method. Background information includes specific facts such as the physical properties of materials, accepted formulae and approximations which can be used when detailed information is not available.

Every inference engine requires a set of rules to guide the execution of its task. A rule is any piece of knowledge upon which decisions can be based. It may be a well investigated and formalized relation, an approximation or a rule of thumb based on experience. Every human expert uses rules of some sort to analyse problems. These rules must be isolated and included in the knowledge base. This task is a difficult but rewarding part of developing an expert system.

iii. User Interface

The user interface is the component of the expert system concerned with data transfer and communication. It consists of all the input/output utilities used to interface with peripheral devices to the communicate with the user. It is used to compile, alter and review the contents of the knowledge base including the problem description, background information, rules and output. It includes the drivers for the keyboard, disk drives, printer and video screen. For convenience of the programme user, these drivers are accessed through editors and graphics utilities that arrange and sort information for easy interpretation.

Although the user interface does not participate directly in the application of expert knowledge, it is a very important aspect of the expert system because it is the portal through which the expert system interacts with the outside world. A well designed user interface enhances the operation of the system while a poorly designed one may effectively cripple it. It must be comfortable and easy to learn, understand and use, yet powerful enough to communicate complex problems and ideas because it is the sole link between the operator and the expert system.

CHAPTER 6

DEVELOPMENT OF A STOCHASTIC COST ESTIMATION METHOD FOR COMPUTER-BASED ANALYSIS

A. Development Goals

The goal of this chapter is to layout the design of an innovative new cost estimation method for use in a computer-based expert system. The development of this method provides an ideal opportunity to assess and improve current methods in light of expert systems technology in order to achieve an optimum balance of method, effort and accuracy. The method will integrate the basics of cost estimation and the principles of computer-aided systems as discussed in previous chapters. Unlike current methods, it will be created specifically for use in a computer environment and be free of the limitations imposed by manual cost estimation methods. It will attempt to exploit the strengths and avoid the weaknesses of computer technology in order to develop a new tool for cost estimation.

B. Summary And Description Of Strategy

The proposed cost estimation method integrates proven methods with new concepts. The core of the system was established using an operations approach which appears best suited for accurate modelling of the fabrication process. It was implemented as an expert system to ensure that the decision making abilities of human estimators

could be fully emulated. In addition, the expert system was written in a modular form that can be easily understood and modified by the end user. Sensitive stochastic analysis was introduced to replace subjective judgement in the estimation process and augment the estimate with detailed cost distribution information. This information can be used to calculate confidence limits which can be used to determine the probable accuracy and reliability of cost information. Such analysis is impractical to perform manually yet provides information which is vital for the comparison of designs and proposals.

Minimization of direct human interface with the expert system during its operation is a key element of the expert system proposal because each interface reduces the effective speed of computer calculation. Calculation speed is an important asset of computer technology which must be fully exploited in order to ensure that the cost estimation system is a practical alternate to the use of human experts. Minimizing human interference also ensures that the expert system executes its task consistently and without bias making it a valuable tool for the comparison of alternate designs.

Standardization is one of the techniques used to minimize direct human access and maximize the power of the user interface. It is a method of conceptualization which is regularly used by human experts. It allows large quantities of information to communicated very effectively. While standardization at first appears to be an unrealistic and overly restrictive imposition on industry, it is not unprecedented. Many engineering and fabrication offices develop inhouse standards. In some countries, notably Australia and those in the European Community, industry wide standards have been adopted. Standards provide a framework in which design can proceed with great efficiency. For example, structural steel sections are only available in certain shapes

and sizes. Although no particular section may meet the requirements of a given structural element exactly, there is usually one or more acceptable sections. Clearly, it is more efficient and convenient to produce large quantities of standard sections rather than rolling them individually for every project.

The driving force behind standardization is the economy. Standard components are generally less expensive to produce because they do not have to be individually detailed and fabricated. In addition, their production can take advantage of any existing economies of scale. Moreover, the adoption of standard components is strongly encouraged by the development of computer-aided systems because standardization is a conceptual tool which increases the efficiency of the user interface and provides reference points for analysis. Standardization is a valuable tool because it not only aids in cost estimation, it actually reduces the cost of design and fabrication.

There are two main uses of standardization in the proposed cost estimation method. Firstly, the method requires a set of standard operations which can be used to describe the fabrication of structural elements. This set must be complete and exhaustive so that every item can be fully described in the cost model. Secondly, a catalogue of standard members and connections is needed so that each project can be described quickly and easily without burdening the user interface with a high level of detail. Each standard element in the catalogue contains a description of the material and operations required for fabrication. The catalogue must be updated and expanded periodically to reflect new construction methods. Without such conventions, the cost of communicating fabrication information would soon outstrip the benefits of computer-based estimation.

C. Evaluation Of General Methods

The selection of a basic estimation method is perhaps the most crucial decision in the design of the new method. It must be detailed enough to justify its implementation in an expert system while allowing the advantages of computer technology to be fully exploited. At the same time, it must be recognized that the number of estimation parameters is limited by practical difficulties in the collection of meaningful cost data. The basic method must optimize its level of detail to guarantee that the maximum accuracy of estimation can be achieved while ensuring that the method remains practical enough for implementation. It is a question of balancing the desire for theoretical accuracy against the problems of implementation and data acquisition. Thus, the creation of a new method for cost estimation provides an opportunity to review the basic methods of cost estimation in light of the abilities of expert system technology.

The method of estimation by characteristic parameter is inherently subjective. It requires a unit cost based on a qualitative subjective assessment of the structural design. Such assessment is difficult to include in an expert system because it is an ill-defined abstract process entirely with in the mind of the estimator. It is awkward to codify because the process is not formally conceptualized and may not be well understood by the estimator himself. In addition, the information required for the computer to emulate subjective decisions would require an impractically large number of parameters. Moreover, the actual calculation algorithm of this method is too simple to justify the use of powerful computer technology. Overall, estimation by characteristic parameter does not seem to be well-suited for use in a computer-aided system.

The method of estimation by units involves a great deal of data manipulation and transfer; however, it still has a significant component of subjective assessment. The manipulation and transfer of data is a task that is well-suited for use in a computer environment, so programmes can be, and have been, written to aid experienced cost estimators with this method of estimation. Such programmes increase efficiency by making optimum use of the estimator's subjective skills; however, they do not challenge the capabilities of modern computer technology. The computer remains idle most of the time awaiting input or instructions from the human estimator. Unfortunately, the unit cost values must still be determined subjectively, so the method is not well-suited for use in an expert system.

The method of estimation by operations relies almost entirely on data manipulation and transfer to complete the estimation process. It models the cost of fabrication directly making it the only method which can accurately predict the cost of new types of structural elements. Because it does not rely on implicit subjective assessments, cost data can be recalled directly from a cost database rather than relying on the input of a human estimator. This minimizes interaction by the operator allowing the computer to execute its programme uninterrupted; therefore, a programme implementing this method uses computer time much more effectively than a programme implementing the method of estimation by units. The method of estimation by operations is the best suited for use in an expert system. Correspondingly, this method was selected for implementation in the proposed expert system.

D. Estimation Process

Once the method of operations was chosen as the basis of the new cost estimation method, it was necessary to establish a specific algorithm for estimation including model parameters and background information. These components were chosen keeping the philosophy of the new estimation method in mind.

At this point, it is necessary to have a conceptual overview of the operation of the proposed cost estimation system [fig. 23]. The programme operator, who is not necessarily an experienced estimator, prepares a set of parameters for the expert system. The expert system takes these parameters and compiles a detailed list of the operations required to fabricate the structure using information from its operations database. It then matches this information with unit cost values from the cost database. Finally, its calculation algorithm computes an estimate of mean cost and variability so that a reliable confidence range of total cost may be determined.

The choice of model parameters is an important decision because it directly affects the project description and the complexity of calculations. Since the goal of the new method is to create an expert system for the use of non-expert users, it was decided to limit the parameters to physical attributes of the structure which could be determined in a direct quantity survey. In other words, all the parameters are physical quantities which may be measured or counted. No measures of quality or complexity were included because this type of parameter assumes a certain expert knowledge on behalf of the programme user.

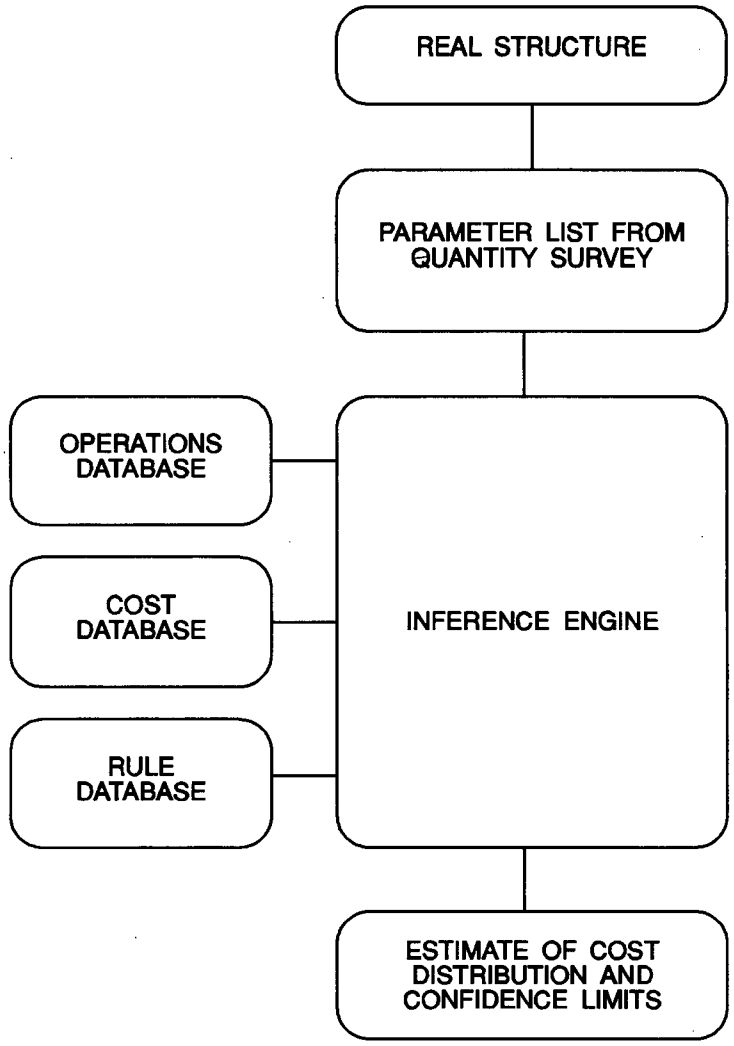


Figure 23: Overview Of Operation

The basic parameters describe the structure in terms of standard members and connections. The description includes a designator which indicates the type of element and one or more modifiers which are required to indicate its size or geometry. For

example, a simply supported beam element might contain modifiers to indicate the size and type of section as well as the connections on each end [fig. 24]. The programme user requires a catalogue of standard members and connections in order to prepare the parameter list.

SSBEAM W310*89 5000 C1 C2

Designator : SSBEAM

Section Modifier : W310*89

Length Modifier : 5000

Connection Modifier : C1 and C2

Figure 24: Designator And Modifier: An Example

The operations database is the first type of background information required by the expert system. Every designator in the parameter list corresponds to a set of instructions in the database which may be used to expand the modifiers into a comprehensive list of operations required to fabricate the designated item. The quantity of each operation is described in terms of one or more parameters used in the cost calculations. There is no theoretical limit to the number of parameters which may be used to describe each operation or the complexity of the cost formulae; however, there are practical limitations. A reliable estimate of each parameter and its associated cost factors must be calculated on the basis of historical data. For reasons of simplicity, a simple linear

model with one cost parameter per operation was chosen for use in the proposed method. The operations and their parameters are discussed further in subsequent chapters. A more complicated method may be introduced in future cost models.

The cost database is the second type of background information required by the expert system. It contains information regarding the cost distribution of each operation. At present, a Gaussian distribution of cost is assumed for ease of calculation so a mean cost and variance are all that is necessary to fully describe each distribution. Future research may consider other possible cost distributions. The acquisition of cost information is discussed in detail in subsequent chapters.

The calculation algorithm combines the background and project data using well known statistical formulae based on least squares to calculate the mean cost and its expected variation [fig. 25]. With the assumption that the unit costs are normally distributed, confidence limits for project cost can be easily calculated. Other distributions may be used; however, they tend to require more complex calculations and no evidence exists to show that they describe cost more accurately. Future research should consider this question in more detail.

The Basic Cost Formulae Is:

$$c = [1 \quad q_1 \quad q_2 \quad q_3 \quad \cdots \quad q_k] \begin{Bmatrix} u_0 \\ u_1 \\ u_2 \\ . \\ . \\ . \\ u_k \end{Bmatrix}$$

Where:

c is the total cost of the project.

$[q]$ is a row matrix of operations.

$\{u\}$ is a column vector of unit costs.

Therefore Using A Least Squares Approach:

$$\hat{c} = [q]\{\hat{u}\}$$

$$\hat{\sigma}_c^2 = \hat{\sigma}^2 [q]^T [[Q]^T [Q]]^{-1} [q]$$

Where:

$\hat{c}$ is the predicted mean cost of the project.

$\{\hat{u}\}$ is a column vector of calculated unit costs.

$\hat{\sigma}_c^2$ is the variation of the predicted project cost.

$\hat{\sigma}^2$ is the variation of observed project costs.

$[Q]$ is the matrix of operations for observations.

See Chapter 9 For A Detailed Description Of Observed Quantities.

Figure 25: Calculation Algorithm

CHAPTER 7

OPERATIONS

A. Definition Of A Working Set

In this thesis, an operation is a well defined task used to prepare, shape or join materials in the fabrication process. Each operation consists of one or more related subactivities which are needed to complete the task. The production of any fabricated item is defined by the operations required.

The operations approach is a method of conceptualizing the fabrication process. It is a framework for both cost estimation and the design of production methods. The approach requires the establishment of a well defined set of operations that describe the available methods of steel fabrication completely and exhaustively. Like any model, the operations approach is only useful and valid when it is able to completely describe the problem being analysed. An incomplete set of operations will result in poor conceptualization and limit the usefulness of the operations approach because some elements of the fabrication problem will be outside the scope of the cost model.

The set of operations used in the model cannot be chosen lightly. It must reflect the philosophy of the operations method while also considering the practicality of implementation. Previous researchers in this field have suggested various sets of operations. The set chosen for use in this model was derived from the set proposed by Y.C. Leung. The fifteen operations chosen by him for work in his thesis were:

a) shearing;

b) sawing;

c) burning;

d) punching;

e) drilling;

f) making templates;

g) fitting;

h) welding;

i) cleaning;

j) painting;

k) handling;

l) machining;

m) preparing shop drawings;

n) laying out; and,

o) others.

There are several problems with the implementation of this set of operations in an expert systems environment because:

a) the set is not truly complete and exhaustive because it includes a general miscellaneous category: other;

b) the operations are not unique because, in some cases, one name may to refer two separate fabrication procedures;

c) the parameters required for some operations require considerable subjective assessment; and,

d) the distinction between operations and subactivities is not well defined.

Although the set of operations is complete in a technical sense, it is not complete and exhaustive because it uses a general miscellaneous category, other, to achieve completeness. This is not an operation because it is not a well defined task and is without any established subactivities. Such an approach can not be used in a computer environment because artificial intelligence is not yet developed to the stage where the computer can independently identify and define a new operation. The expert system requires an explicit set of operations which is capable of completely describing every fabrication procedure. In addition, provisions should be made for editing and expansion of the knowledge base to accommodate new operations as they become available. In other words, the expert system must be capable of learning new operations although the identification and definition of such operations is left to the user.

The uniqueness of operations is not truly a necessity but it is an important convenience which increases clarity and enhances communication. Ideally, each conceptual operation should be associated with one and only one fabrication activity. Some of the operations proposed by Leung refer to activities involving different materials and equipment with very different unit costs. Clearly, these activities are separate operations. Accordingly, three of Leung's operations were reconstructed. Burning was sub-divided and renamed to become flame cutting - plates and flame cutting - shapes. Likewise, punching was divided into an operation for plates and an operation for shapes. Cleaning was also separated into two new operations: wheelabration and sandblasting. These revisions ensure conceptual and semantic clarity while allowing

the cost of each operation to be modelled with greater accuracy. Note that there may be a need for further subdivision of operations particularly the operations describing painting and welding.

The practicality of implementation is always an important consideration in the design of methods for use in an expert system. It is important to keep the needs of the end user in focus. Some of the operations proposed by Leung such as fitting, lay out and making templates require subjective judgement and interpretation. This implies that the users have expert knowledge and defeats the argument that expert systems reduce the need for expert consultation. Such conflict can be avoided by redefining the operations set so that all the required parameters are quantitative rather than qualitative. Then, the appropriate quantities may be determined directly from a quantity survey, thereby eliminating all subjective parameters. This requirement implies that each operation must physically alter some component being fabricated guarantying that each operation has a tangible indicator which can be used as a cost parameter. It also excludes operations such as handling and preparing shop drawing which are necessary but do not directly alter any material.

The distinction between operations and subactivities is largely a matter of definition; however, the distinction is important because it is necessary to ensure that each sub-activity is included once and only once in the cost model. There should be no confusion as to whether an activity is an operation or subactivity. Some of Leung's activities such as handling are too general because they are both operations and subactivities. Other tasks, such as clean-up, are not considered at all. To prevent

confusion, the proposed method stands by its principle of using only physical parameters for estimation. Activities such as handling and clean-up are attributes of larger operations because they do not directly alter material.

After careful consideration, fifteen operations were chosen and defined as a working set. These operations are:

a) shearing;
b) sawing;
c) flame cutting - plates;
d) flame cutting - shapes;
e) punching - plates;
f) punching - shapes;
g) drilling - plates;
h) drilling - shapes;
i) bolting
j) welding;
k) cleaning - wheelabration;
l) cleaning - sandblasting;
m) painting;
n) galvanizing; and,
o) machining.

This set of operations will probably require further adjustment and refinement before it can be established in a working industrial or engineering environment. The actual

set included in any expert system will vary because the capabilities of each fabrication shop are different. For example, galvanizing is an operation which is often used to protect steel against corrosion, but it requires specialized equipment which is not available in every shop. In general, the more operations in the set, the more accurately cost can be modelled. The set must also be modified as different operations and equipment are developed and become available.

B. Unit Cost Of Operations

Many different methods and strategies can be used to assess cost. These methods are all based on some rationale that is intended to reflect the accrual of cost; however, the actual accrual is a random function that is highly dependent on the complex interaction of material, labour and equipment. It is influenced by corporate policy, market conditions and desired profits. In fact, cost is difficult to measure exactly even after all work has been completed. The accrual of cost is certainly not a simple function. Nonetheless, simple methods are often used to estimate it. At least these methods provide some means of predicting and controlling cost.

The most accepted methods of estimating are the unit cost methods. Unit costs represent the mean cost of completing a single unit of production. These methods usually assume that costs are linear functions involving quantities of the operations as measured by some parameters and mathematical constants referred to as unit costs. This assumption greatly simplifies prediction because it fixes the form of the equations

used to predict the cost of each operation. It also ensures that the principle of superposition is valid and that fabrication activities may occur in any order. It is important to remember that this assumption is for convenience only and does not necessarily reflect the realities of steel fabrication.

In many unit cost methods including the one proposed by Leung, costs are evaluated per unit time requiring an estimate of the productivity of each subactivity. This approach introduces unnecessary uncertainty and calculations into the estimation problem because the estimation of productivity is an unnecessary burden which can be avoided. In the proposed method, the unit costs are evaluated per unit operation directly. This approach is especially convenient because it minimizes the number of unknowns in the estimation problem and reduces the need for subjective judgement. The unit costs for the proposed working set of operations can be derived directly from shop records using cost and production information from previous projects to establish a best fit curve. Some adjustment of these values may be necessary to account for predictable changes in unit costs due to known changes in the cost of labour, equipment and materials, but no productivity information is required.

Unit cost methods are only tools used in the prediction of cost. Actual costs depend on many factors including the sequencing of operations which can not be assessed without a direct simulation of shop conditions. The unit cost values reflect the typical sequencing of a particular fabrication shop. Costs may be reduced by more skillful sequencing and/or optimization using a simulation programme. Such techniques are beyond the scope of this thesis.

C. Variability Of Unit Costs

Traditional approaches to cost estimation consider only the mean cost of fabrication and ignore the uncertainty of the cost function. Historically, the variation of costs was ignored because it was impractical to calculate; however, the required technology is now available. Statistical techniques have been developed which make calculation of the variation of project cost possible, and computers are available which make the introduction of these techniques practical.

Uncertainty exists in almost all aspects of cost estimation. The most certain element is the amount of physical work to be done because this can be calculated in a quantity survey. Unit costs are the most variable element because they depend on the rate and efficiency of production which can be highly variable. They are primarily affected by the complexity and scheduling of fabrication; however, they can also be affected by many unanticipated events such as the failure of equipment, the availability of skilled labour and the interaction of projects within the shop. These random events may disrupt production and drive up costs significantly. In the interests of developing a useable model, all variability is assumed to occur within the unit cost variable because this variable is responsible for most of the variation in cost. Quantities are assumed to be constant and correct because of the considerable difficulties involved in predicting their variation. Also, the cross-correlation of costs due to scheduling interactions was ignored. These assumptions are normal engineering approximations which reduce the complexity of the problem so that it may be easily solved.

The variation and covariation of unit costs can be assessed numerically at the same time as the mean unit costs are calculated from production records. This data may be applied in the cost model to predict the mean and variability of project cost allowing the range of probable costs to be established.

D. Specific Operations

The working set of operations is described below. This information represents background information on the steel fabrication industry which must be included in an expert system for cost estimation.

i. Shearing

Shearing is the act of cutting metal using hydraulic shears. It is the preferred method of cutting smaller bars, plates and channels. The major advantages of shearing over other methods of cutting is that it is a fast, clean method which can be used to cut bundles of identical components simultaneously.

The sub-activities of shearing are set-up, stock movement, shearing and clean-up. These sub-activities are typical of many operations. First, the equipment must be adjusted and supplied with raw stock. The stock may then be moved by a conveyor

and sheared into appropriate lengths as required. Finally, any waste material must be disposed of. The key parameter of shearing is the length of material which must pass through the jaws of the shears. This parameter indicates the amount of time required for the operation and limits its productivity. Thus, the unit cost of shearing is governed by the movement operation and is expressed as a cost per linear length. The sub-activity of set-up depends on the type of equipment used. Movement depends on the speed of the conveyor system which is largely independent of stock size. The sub-activity of shearing itself represents only a small fraction of the time and effort required to perform the shearing operation and is fairly constant regardless of the sheared cross-sectional area. Clean-up is simple and consists only of discarding the unused ends of stock. Shearing identical pieces simultaneously reduces the effective length of material passing through the jaws of the shears and reduces cost. The effective length of material being sheared must be determined by an expert evaluation of the number of pieces that can be sheared simultaneously and the length of raw stock.

ii. Sawing

Sawing is the act of cutting metal using a rotary blade saw. It is an alternate method of cutting which is preferred to shearing when larger shapes or stacks of bars must be cut. Like shearing, it is a fast method that produces a clean cut; however, it is slower than shearing and produces more waste. On the other hand, the capacity of a metal saw is much larger than that of shears.

The sub-activities of sawing are set-up, stock movement, sawing and clean-up. It can be debated whether the sub-activity of movement or sawing governs this operation; however, assuming that lighter gauge elements are sheared, the sub-activity of sawing should govern. Thus, the cost of sawing can be expressed as the cost per cross-sectional area or weight per unit length. (The weight per unit length being directly proportional to cross-sectional area.) Set-up and movement activities are similar to those of the shearing operation. The speed of the sawing sub-activity depends on the cross-sectional area of the material being sawn. Clean-up is more costly than that of shearing because the waste material removed by the saw must be discarded. Note that the volume of waste material is also proportional to the cross-section of the stock.

iii. Flame Cutting - Plates

Flame cutting uses a jet of oxygen from an oxyacetylene cutting torch to blow away molten metal in order to cut through steel. It is the preferred method of cutting thick plates which can not be sheared or any plates which have a complicated shape. It may be performed manually or automatically with or without templates. It is an extremely efficient means of fabricating plate elements such as gussets, webs and flanges.

The sub-activities required to flame cut plates are positioning, starting, cutting and clean-up. Generally cutting is the most significant sub-activity. The volume of metal which must be removed is a good indicator of the quantity of cutting required because it indicates the amount of metal which must be melted and blown clear. This can be

estimated once the thickness of the plate and length of the cut are known. Thus, the cost of flame cutting plates can be expressed as a cost per cross-sectional area of cut. The positioning of plates is much simpler than the other cutting methods discussed so far because no movement is required. The start of a cut involves some delay as an initial pool of metal must be melted. After that, the actual cutting process proceeds at a constant rate. Finally, waste metal must be collected and discarded. The efficiency of this method can be increased by cutting identical shapes in stacks.

iv. Flame Cutting - Shapes

The flame cutting of shapes relies on the same physical principles as the cutting of plates; however, it is a distinct operation. Unlike the flame cutting of plates, it is usually performed manually. It is generally slower, more expensive and requires more dexterity; however, it is one of the most versatile fabrication operations. In a small fabrication shop, flame cutting equipment can be used to perform many operations. It can be used to cut, notch, coup and produce holes as required.

The sub-activities of flame cutting shapes are identical to those of flame cutting plates. Only the economics changes because the flame cutting of three dimension shapes is more complex than the cutting of two dimensional plates. The cost of flame cutting is governed by the amount of metal which must be removed and is estimated by the cross-sectional area which must be cut. This operation is likely to have a high variability

simply because it can be used for so many purposes. In some shops, it may be advisable to separate this operation into more specific operations so that cost can be modelled more accurately.

v. Punching - Plates

Holes may be punched in plate material by using hydraulic or mechanical punches. They may be made individually or in groups using a multiple punch. Punching is the preferred method of creating holes because it is faster and less extensive than drilling; however, the thickness of the plate and aspect ratio of the holes must be suitable for this operation to be applied. Like many plate operations, the efficiency of this operation can be increased by stacking plates.

The sub-activities of this operation are set-up, positioning, punching and clean-up. The number of separate applications of the punch is a good parameter upon which to base the cost of punching. Unfortunately, this parameter is slightly more difficult to determine than other the parameters discussed because it is not a part of a traditional quantity survey. Some expert knowledge must be provided within the expert system to determine this parameter. One method is to assume that all holes on a given connection can be made by one application of the multiple punch provided the punch has sufficient capacity. The cost of set-up is a function of the type of equipment being used. Positioning is similar to the positioning of flame cut plates. Positioning and repositioning the plate for each punch represents a significant cost. As in the case of

shearing, the actual sub-activity of punching is probably independent of the size of holes. Punching is a fairly clean operation producing only a modest quantity of waste that can be handled easily.

vi. Punching - Shapes

Generally, the holes in shapes are produced individually using a detail punch. Again, punching is the preferred method of creating holes whenever the shape material and size of hole are within the capabilities of the available punching equipment. Unlike the punching of plates, shapes can not be stacked to improve efficiency.

The sub-activities of punching shapes are set-up, positioning, punching and clean-up. The operation is more akin to shearing than the punching of plates because each element must be moved along a conveyor and positioned for punching. Thus, the cost of punching shapes is expressed as a cost per linear length.

vii. Drilling - Plates

Drills are used to produce holes in plates when punching is unsatisfactory or unavailable. Holes may be drilled individually or in groups. Drilling provides a clean

hole, but is slower, more expensive and less convenient than punching. Drilling and punching are related in much the same way as shearing and sawing are. Again, the efficiency of the operation can be increased if plates are stacked.

The drilling operation requires set-up, alignment, centre punching, drilling and clean-up. In addition, a pilot hole may be required for larger holes. The cost of drilling is related to the amount of material which must be removed. This quantity can be estimated from the number of holes, plate thickness and size of holes. Drilling thickness is a good estimator of cost in typical fabrication because most holes are drilled for standard sizes of bolts. Thus, the cost of drilling in typical fabrication can be expressed as a cost per total drilling thickness. The cost of set-up, alignment and centre punching depends on the type of equipment being used. The cost of drilling itself depends on the amount of material being removed. Clean-up costs are higher than those of punching because the waste material is less convenient being bulky and sharp. The cost of clean-up is proportional to the volume of metal removed.

viii. Drilling - Shapes

The drilling of shapes is, in principle, the same as the drilling of plates; however, it is more complicated and more costly because shapes are three dimensional while plates are two dimensional. The drilling of shapes is often done on a conveyor line like the punching of shapes.

The sub-activities of drilling shapes are set-up, movement, alignment, centre punching, drilling and clean-up. The cost of this operation may be governed by the number of holes or by the time it takes to move items on the conveyor system. If it is assumed that the holes are unsuitable for punching, then it is probable that holes must be carefully aligned or drilled through thick material. In either case, cost will probably be governed by the drilling sub-activity rather than the movement. Thus, the unit cost of drilling shapes is expressed as a cost per drilling thickness. Otherwise, the sub-activities of drilling shapes are similar to those of drilling plates.

ix. Bolting

Bolting is a common means of connecting fabricated items using mechanical fasteners. The sub-activities of bolting are fitting and bolting. Fitting is the positioning of sub-assemblies including whatever temporary connectors are required to hold the assemblies in place. Bolting is the attachment and tightening of all the fasteners required to carry the design load. The operation is difficult to assess quantitatively but the number of bolts required is probably a good estimator of cost because the complexity of fitting and the labour requirement increases as the number of bolts increase. The unit cost of bolting can be expressed as a cost per bolt.

x. Welding

Welding is a means of connection in which items are joined by the fusing of metal. It is an important process in modern fabrication. There are many different types of welding and many types of welds. The operation should probably be divided into further separate operations that reflect the diversity of welding activities.

The sub-activities of welding are preparing, positioning, preheating, welding, finishing and clean-up. The volume of weld material required is a key estimator of cost. Larger volumes indicate that the weld is longer or requires more passes to complete. Volume can be determined from the size, length and type of weld. Thus, the cost of welding may be expressed as a cost per volume. The type of preparation required for a weld depends on the type of weld and is generally a function of the weld length. Preparation consists of grinding, cleaning and fitting. Positioning time, and hence cost, also increases as weld size increases because the components tend to become larger and must be positioned more accurately. Preheating is not normally required under shop conditions. Welding is a function of the volume of weld material which must be deposited. Finishing of the weld including chipping and grinding is a proportional to weld length. Finally, clean-up consists of removing slag and ground metal.

xi. Cleaning - Wheelabration

A wheelabrator is a series of rotating brushes used to scrape surface contaminants such as rust and paint off the surface of metal. Wheelabration is the preferred method of cleaning long structural members.

The operation of wheelabration is simple having only one subactivity: wheelabration itself. The entire member must pass through the wheelabration unit on a conveyor belt. Sometimes, each side of the member must be processed separately. In either case, the length of the member is the best indicator of the cost of wheelabration. Thus, the cost of wheelabration can be expressed as a cost per linear length.

xii. Cleaning - Sandblasting

Sandblasting is an alternate method of cleaning which is used for members that are too large or otherwise unsuited for wheelabration. Sandblasting uses a jet of air to propel grit or pellets against the component and remove the surface metal along with any contaminants. This operation may be required to prepare stock for other fabrication operations such as welding, painting or galvanizing. It may also be done for purely aesthetic reasons.

The sub-activities of sandblasting are set-up, sandblasting and clean-up. Cost is largely influenced by the quantity and quality of grit which must be sprayed. This is determined

by the degree of surface contamination and the size of area which must be cleaned. Set-up is simple compared to many fabrication operations because careful positioning is not required; however, the component must isolated to protect workers and equipment from flying grit. The sub-activity of sandblasting itself requires the movement of the sandblasting jet over the entire surface of the item. Finally, the sandblasting grit and the material removed from the metal surface must be collected. Disposal of this waste may be expensive as it may be considered to be hazardous waste. The surface area to be cleaned is a good estimator of both the sandblasting and clean-up activities. Thus, cost may be expressed per unit area to be cleaned.

xiii. Painting

Painting is one of several methods which can be used to protect steel from the elements. It provides a barrier against corrosion and can also serve as a aesthetic finish. Many types of paint exist and more than one coat of paint may be required. Like welding, the operation of painting may require further sub-division in order to fully describe the options available to the steel fabricator.

The sub-activities of painting are set-up, application, drying and clean-up. The set-up is similar to sandblasting because the item must be isolated to protect workers and equipment from paint and fumes. During application, the surface of the item is covered with a coat of paint using a jet or brush. Drying time is then required to allow each coat to set and harden. Drying may govern the capacity of painting equipment. Finally,

paint residue must be removed from equipment. Note that clean-up may occur during in parallel with the drying sub-activity. The cost of painting is dependent on the area to be protected. Thus, the cost of painting may be expressed as a cost per surface unit area. An effective area must be established which reflects each coat of paint.

xiv. Galvanizing

Galvanizing is the coating of an item with a thin layer of zinc to provide cathodic protection against corrosion. While it is an extremely effective means of protecting against the elements, it does not provide an aesthetic finish. Although there are several methods of applying the zinc coating, galvanizing in the fabrication industry is almost always accomplished by submersion in a bath of molten zinc.

The sub-activities of galvanizing are cleaning, pickling and dipping. Each of these activities involves the immersion of the item in an appropriate bath. Cleaning and pickling remove rust, scale and surface contaminants to prepare the item so that it will alloy evenly with the zinc in the final dipping. The cost of galvanizing is probably dependent on the number of items rather than any size parameter because the cost of operating a galvanizing facility is incurred continuously. Each bath must be maintained within certain quality parameters such as acidity, oil content and temperature. The cost of the zinc itself is minor and the same equipment is usually used to handle the items regardless of size. Each item must be dipped for approximately the same amount

of time, although smaller items may sometimes be dipped together. Thus, the cost of galvanizing may be expressed as a cost per item. Note, however, that it is also popular to assess the cost of galvanizing per unit area.

xv. Machining

Machining is the process of shaping metal to fine tolerances using lathes or milling machines. Although it is the mainstay of mechanical fabrication, it is less commonly used in structural fabrication. It is only used when a close fit of elements is required such as in bearing connections where the load is transferred directly from one member to another without reliance on fasteners.

The sub-activities of machining are positioning, machining and clean-up. Careful positioning is required to ensure that fine tolerances can be met. The actual machining activity uses a bit or router to cut away unwanted material. This material must then be collected and disposed of. It is sharp like the waste from the drilling operation and must be handled carefully. Although machining removes a volume of metal, it is probably best estimated on the basis of surface area because the thickness to be removed is often uncertain. Thus, the cost of machining can be expressed as a cost per unit area.

CHAPTER 8

DEVELOPMENT OF THE EXPERT SYSTEM SHELL

A. Development Of An Expert System Shell

The development of the expert system shell is separate and distinct from the development of the cost estimation system. The shell is a tool for manipulating general knowledge. It does not become a complete expert system until it is linked to an appropriate knowledge base. The basic goal in the development of the shell is to create a decision making programme capable of efficient data manipulation while ensuring that it is easy to learn, understand and modify. This goal is accomplished in part by including as many familiar elements within the shell as possible. Thus, the development has relied upon established data formats and common utilities written in accepted programming languages. In addition, a natural language control system and parser was developed to reduce the burden of learning complex syntax.

B. Environment

The selection of a suitable programming environment is vital to the development of an expert system. Both hardware and software must be carefully selected to ensure that they can be easily integrated. Moreover, the chosen environment must be compatible with industry standards because an otherwise well developed system is useless if it does not meet the needs of its users.

i. Hardware Environment

The shell must be developed for use on hardware that is acceptable to industry. Many expert system shells have been developed at universities on mainframe computers; however, such equipment is beyond the means of most designers and fabricators. A useful expert system tool must be useable on a machine that is available to and affordable by industry. In addition, it is advisable to select hardware with a large existing software library to ensure that the development of the shell is not hindered by a lack of suitable utilities and supporting software.

The selection of a basic computer system was the most important hardware decision. Various computer manufacturers produce machines of several types and sizes to different standards. While the proponents of these systems expound on the virtues of their preferred machines, the real question is which machine or machines are accepted by industry. A telephone survey of designers and fabricators in the Vancouver area revealed that the overwhelming majority of machines in industry are personal computers compatible with IBM standards. Apple machines were a distant second with a few other miscellaneous machines adding colour to the survey. Only a few major companies had access to mainframe machines and these firms also relied heavily on personal computers. In order to be compatible with industry, the expert system shell was designed to operate on an IBM compatible personal computer. The popularity of these computers appears to be due to their high power, low cost and the large amount of available software.

The actual machine selected for development was an IBM compatible Compaq SLT/286. This machine was equipped with 640K RAM, 30M hard drive, 1.44M floppy drive and a Intel 80286 microprocessor. Similar machines are common in industry. While the performance of the expert system is limited by the speed and memory capacity of this particular machine, the shell was designed to be portable and can be expanded to take advantage of the increased speed and memory capacity of other machines. Although the shell will operate on slower machines with less memory, the Compaq probably represents the minimum requirements for reasonable use of the expert system.

ii. Software Environment

While the selection of hardware governs the library of available software, the selection of software is still a critical decision. Software dictates the programming language, data format, communication protocol and appearance of the expert system shell. Careful selection minimizes the programming burden and enhances the operation of the shell.

The selection of a programming language governs the methods which can be used to create an expert system shell. A wide variety of languages are available. Some, like FORTRAN, are older languages which may be preferred because they are traditional programming languages in which many engineering programmes have been written. Others, like LISP and MODULA, have been used in the development of experimental

expert systems. The language selected for the shell must be a powerful well developed language capable of expressing engineering formulae and performing expert system operations. Like the hardware, it should be commonly accepted in industry so that the shell can be easily modified and maintained. For these reasons, the C programming language was chosen for the development of the expert system shell.

A decision to use a spreadsheet environment was made early on in the development of the system shell. This type of environment is familiar to many engineers and facilitates learning and use of the expert system. It can be easily integrated with the C programming language using existing libraries. In addition, it reduces the programming burden by establishing a format for data and providing all user interface utilities including file conversion. A spreadsheet environment also provides a clean visual presentation which enhances the appearance of the expert system. A Lotus compatible spreadsheet was chosen and integrated using the WKS Library. Unfortunately, certain inadequacies in this approach were latter discovered. The sequential data format of the Lotus type spreadsheet reduced the efficiency of the system and limited the number of components that could be included in the estimate.

C. Global Organization Of The Expert System

The expert system has a complex global organization. Its operation depends on the interaction of the shell with the knowledge base. An appreciation of the flow of programme control and information assists in understanding the operation of the expert system [fig. 26].

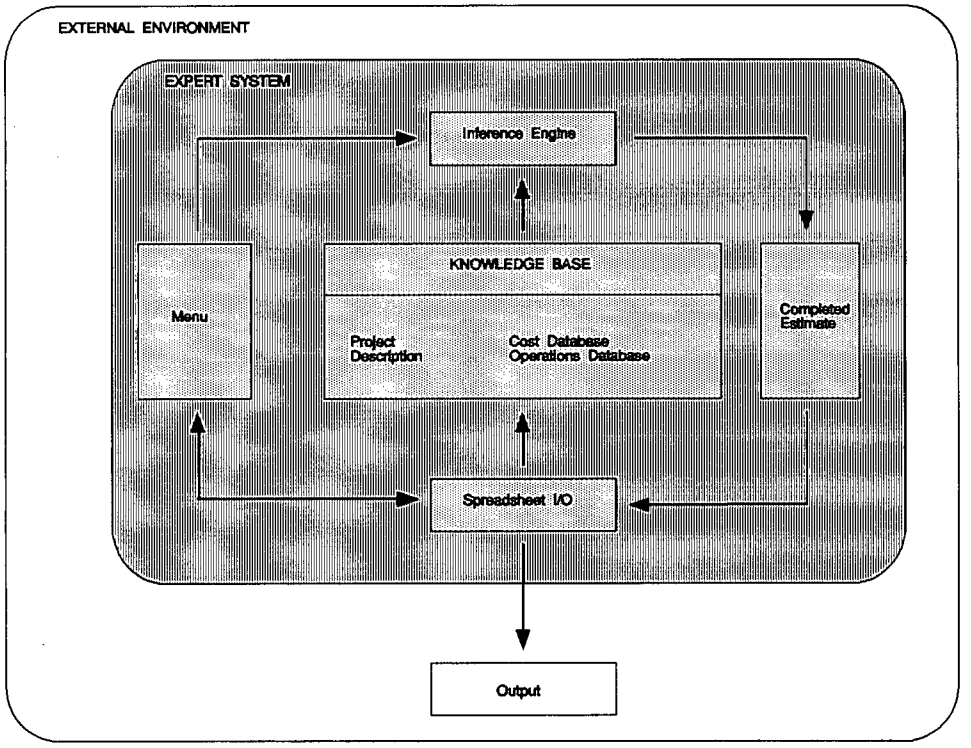


Figure 26: Hierarchy Of Expert System

Access to all components of the expert system is controlled through a small front-end programme. This programme takes the form of a menu through which the operator can update system information, access the knowledge base and initiate the inference engine. In addition, it defines the global variables required to locate programmes and information in the computer's directory structure. Separate data directories may be used to isolate projects or separate cost information from different fabrication shops.

The inference engine is initiated from the front-end programme. It has direct access to the knowledge base and controls operation of the expert system by executing instructions from its rule base using a parser. It has the ability to manipulate data, alter files and branch to make decisions.

The knowledge base consists of two types of files: text and spreadsheet. The rule base is a text file containing a hierarchal set of instructions which are executed by the inference engine. Spreadsheet files contain the data required to describe the problem as well as background information required by the expert system. Each spreadsheet file has a title and headers for each column. Information may processed by row, column or by identifying a particular element of a particular column. For example, an item in column SET may be identified by specifying an item MARKER in column KEY [fig. 27].

The user interface is also accessed directly from the front-end programme. It is composed of text and spreadsheet editors which are used to create files in the knowledge base. The same editors are used to display, alter and print-out all files. These editors were not written specifically for the expert system and, in fact, the expert

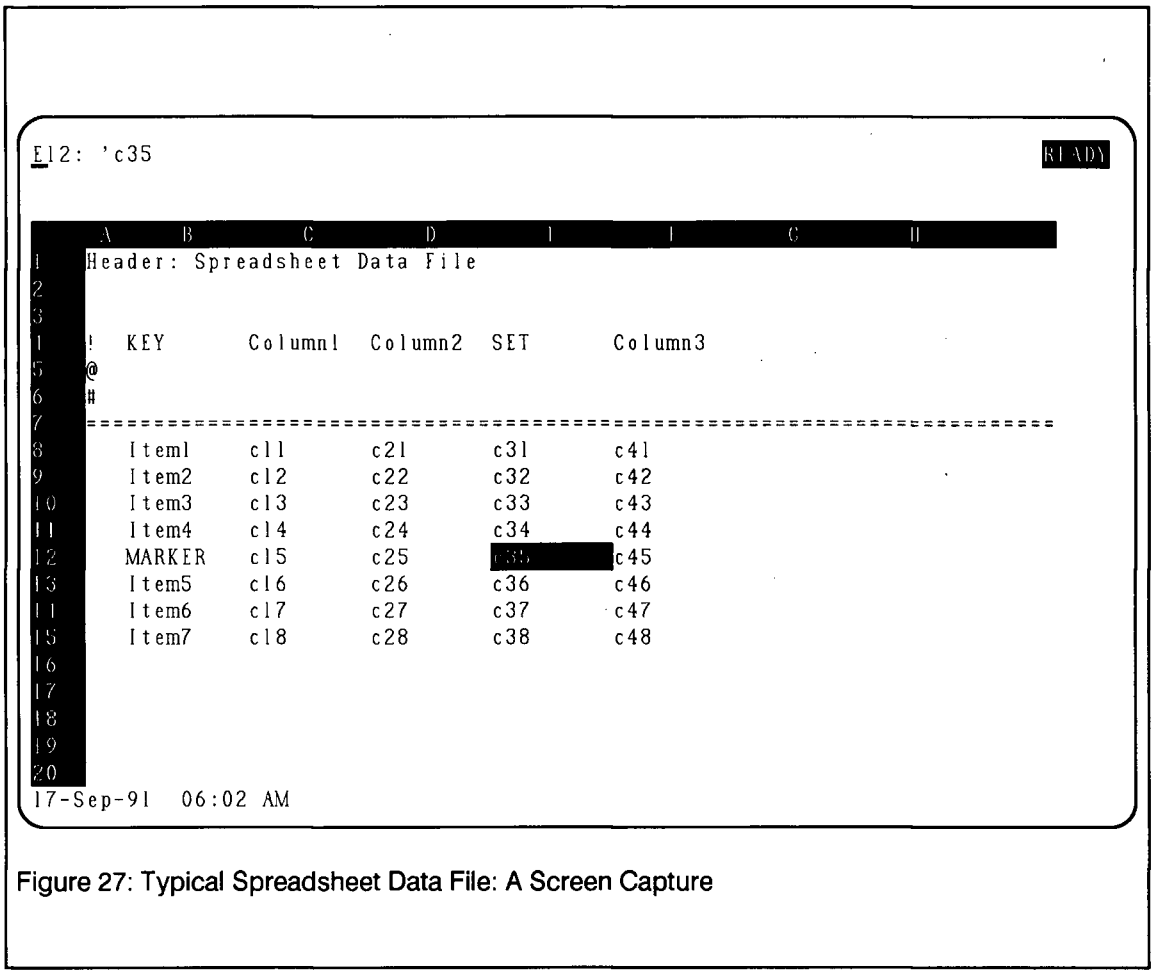


Figure 27: Typical Spreadsheet Data File: A Screen Capture

system was configured so that these elements can be easily replaced. The Norton text editor was used for text files and a generic Lotus compatible spreadsheet was used for spreadsheet editing. The use of these utilities minimized the programming burden.

D. Syntax Of Rule Base Instructions

The rule base is composed of instructions written in a unique programming language. It has a small but powerful vocabulary allowing it to manipulate data and execute controlled branches. The syntax of its instructions is simple and closely follows natural language in order to minimize learning difficulties and enhance readability.

The basic format of each instruction is the same. It has two parts, a command and a predicate. Four types of commands exist:

a) the remark - rem;
b) the subroutine command - call;
c) the positive branch - if; and,
d) the negative branch - ifnot.

The remark command [fig. 28a] is the simplest. It provides commentary and is included solely for the benefit of the user. The inference engine ignores the predicate and proceeds to the next line of instructions.

The call command [fig. 28b] is the most basic instruction that is actually executed by the inference engine. This command is used to order the computer to execute a task when no branching is required. The first part of the predicate is parsed to determine which subroutine must be implemented. The remainder contains instruction which are passed to the subroutine. Each subroutine returns a flag indicating whether or not it

COMMAND :	**REM**
SYNTAX :	**REM COMMENTARY**
EXAMPLE :	**REM THIS IS A COMMENTARY REMARK**

Figure 28a: The Remark Command

was successfully completed. If it was, the next instruction is executed; otherwise, execution of the rule base is terminated. A typical example of a call command is the creation of a new data file or the transfer of data from one file to another.

COMMAND :	**CALL**
SYNTAX :	**CALL SUBROUTINE [DATA1 DATA2 ...]**
EXAMPLE :	**CALL COPY_FILE OLD.WK1 NEW.WK1**

Figure 28b: The Call Command

The branching commands [figs. 28c&d] are similar to the call commands and have the same basic elements; however, the flag returned by the subroutine indicates true, false or failure. Failure flags cause the inference engine to terminate execution. The other

flags determine the next instruction to be executed. The negative branch is not theoretically required because the same instructions can always be written using only the positive branch; however, the negative branch often allows more concise instructions to be written.

COMMAND :	**IF**
SYNTAX :	**IF SUBROUTINE [DATA1 DATA2 ...]**
EXAMPLE :	**IF EXISTS_FILE OLD_FILE.DAT** **CALL DO_IF_TRUE** **CALL DO_IF_FALSE**

Figure 28c: The If Command

E. Hierarchy Of Rule Base Instructions

The thought processes of every human expert can be decomposed into a number of activities and decisions. A decision tree is perhaps the best tool for conceptualizing such an expert method. Unfortunately, decision trees can not be stored easily in either a text or spreadsheet format. Consequently, another method of indicating the hierarchy of decisions had to be devised.

COMMAND :	**IFNOT**
SYNTAX :	**IFNOT SUBROUTINE [DATA1 DATA2 ...]**
EXAMPLE :	**IFNOT EXISTS_FILE OLD_FILE.DAT** **CALL DO_IFNOT_TRUE** **CALL DO_IFNOT_FALSE**

Figure 28d: The Ifnot Command

A text file was established in which the degree of indentation indicates the level of hierarchy. This format is ideal because can be easily read by the user [fig. 29] or converted into a branched linked list [fig. 30] for use by the inference engine. Of course, the file can also be described using a conventional decision tree [fig. 31]. Every time a branching statement is found to be true, the instructions subordinate to the branching statement are executed. If the branching statement is found to be false, the next instruction of equal or higher hierarchal value is executed. If no branching statement is encountered, the next instruction is executed. This is usually on the same hierachal level; however, it may be on a lower level to ensure clarity of visual presentation. When no more instructions are found on or below a particular hierarchal level, the inference engine must back track to determine the next instruction at the level of previous branches. Execution is complete when no more instructions are found.

```
REM A
CALL B
CALL_IF C
 CALL D
 CALL_IF E
  CALL F
  CALL_IFNOT G
   CALL H
 CALL I
CALL J
 CALL K
  CALL L
```

Figure 29: Rule Format As A Text File

RULE : PREVIOUS: NEXT : BRANCH :	A START B NULL	RULE : PREVIOUS: NEXT : BRANCH :	B A C NULL	RULE : PREVIOUS: NEXT : BRANCH :	C B J D
RULE : PREVIOUS: NEXT : BRANCH :	D C E NULL	RULE : PREVIOUS: NEXT : BRANCH :	E D NULL F	RULE : PREVIOUS: NEXT : BRANCH :	F E G NULL
RULE : PREVIOUS: NEXT : BRANCH :	G F I H	RULE : PREVIOUS: NEXT : BRANCH :	H G NULL NULL	RULE : PREVIOUS: NEXT : BRANCH :	I G NULL NULL
RULE : PREVIOUS: NEXT : BRANCH :	J C END K	RULE : PREVIOUS: NEXT : BRANCH :	K J NULL L	RULE : PREVIOUS: NEXT : BRANCH :	L K NULL NULL

Figure 30: Rule Format As A Branched Linked List

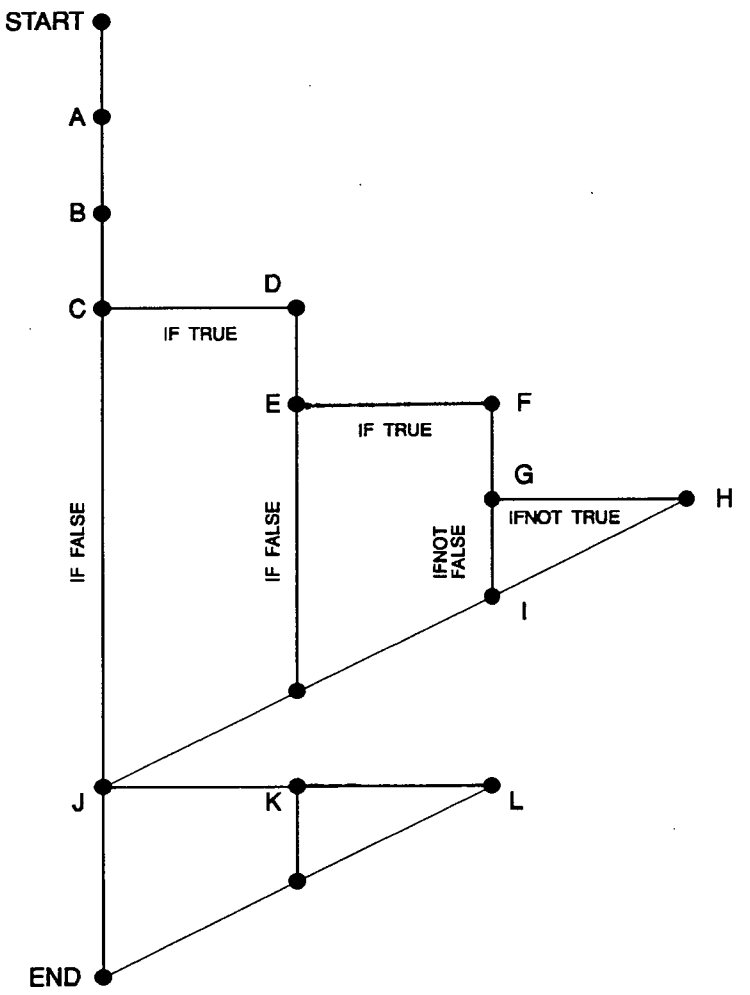


Figure 31: Rule Format As A Decision Tree

CHAPTER 9

KNOWLEDGE BASE ACQUISITION AND MAINTENANCE

A. Importance

The effectiveness of any expert system is highly dependent on the quality of information contained in the knowledge base because the system needs rules and background information to provide a solid basis for its decisions; therefore, it is vital that information be gathered systematically and stored efficiently. In addition, the knowledge base must be maintained by periodic revisions to ensure that its information remains current.

B. Difficulties In Acquiring And Maintaining Knowledge

The acquisition and maintenance of information in an expert system is a major problem which must be addressed before the system can be fully functional. The problem may be divided into five separate areas:

a) identifying sources;

b) extracting information;

c) formatting and storage;

d) processing conflicting strategies; and,

e) selecting updating criteria.

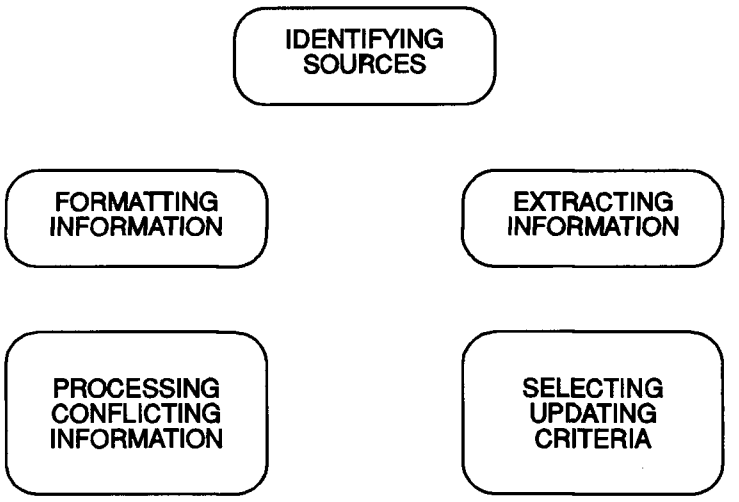


Figure 32: Problems Of Acquisition And Maintenance

i. Identifying Sources

The identification of information sources is the first major stumbling block in the acquisition of information. It is necessary to identify the required information and then seek the most appropriate sources. In order to aid in this process, three classes of information may be established:

a) specific facts;

b) statistical information; and,

c) expert knowledge.

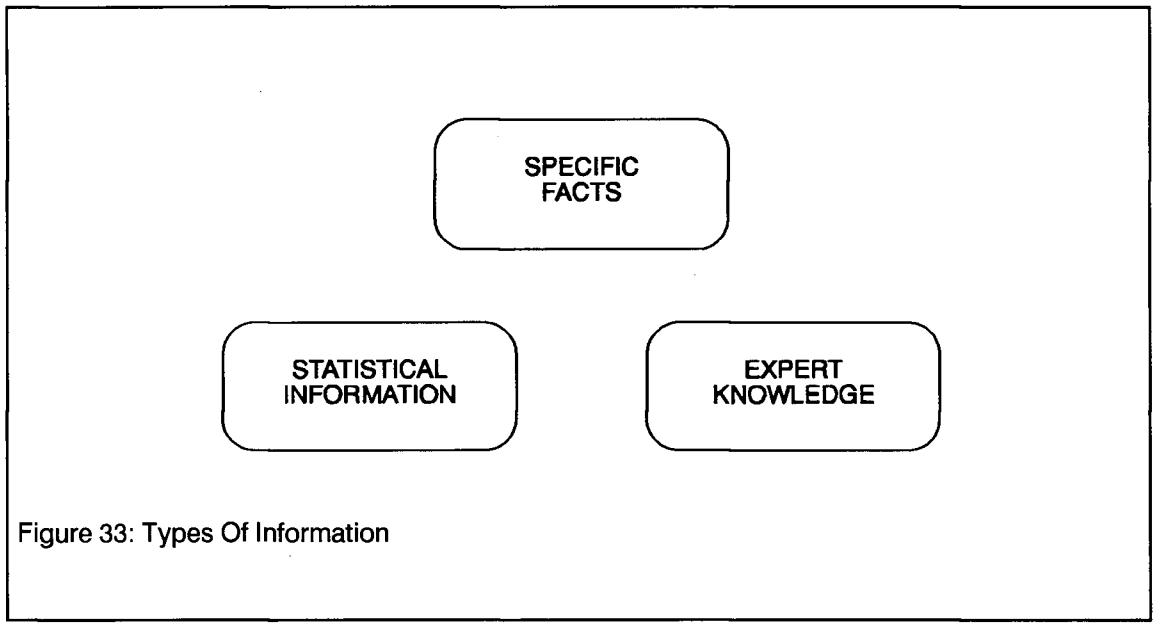


Figure 33: Types Of Information

Specific facts are composed of exact and unquestionable information. They are not subject to interpretation or significant observational error. Industry standards, physical constants and available resources are examples of specific facts which can be located in catalogues, scientific or industrial publications. This type of knowledge can be located easily and while it is essential as background information for an expert system, it is not in itself a decision making mechanism.

Many specific facts are required by an expert system for the fabrication of steel structures including material specifications, design standards and the properties of available steel sections. The Handbook Of Steel Construction published by the Canadian Institute Of Steel Construction is an prime example of a source of specific facts.

Statistical information is composed of observed data which is variable or subject to significant observational error. This type of information is more difficult to obtain and

usually requires some degree of processing before it is useable. Like specific facts, statistical information can be found in scientific or industrial publications, but it is also found in private records which may be proprietary in nature. It is this type of data that is usually required to transform a theoretical cost estimation concept into a useable method. A practical expert system must contain facilities for incorporating such information into the knowledge base when it is available because it is unlikely to be released into the public domain for access by the system developer.

In the expert system for cost estimation, the unit costs and quantities of each operation are statistical data which must be acquired before effective cost estimates can be made. Quantities can be determined by conventional quantity surveys. Unit costs can be derived from the proprietary records kept by each fabrication shop using numerical and statistical techniques. Note that each shop keeps slightly different records and that an effective expert system must be flexible enough to accept information in a variety of formats.

Expert knowledge consists of the rules used by human experts to perform their tasks. This information is most difficult to acquire because the knowledge is not codified and each expert uses his own methods and rules; moreover, experts may be reluctant to part with knowledge that is a cornerstone of their livelihood. They may also have access to confidential information that they can not divulge to outside sources. The identification of experts who are willing and able to contribute to the development of an expert system is a difficult problem, but it is essential that such experts are located because their knowledge is the most important part the system. Once again, the

significance of proprietary knowledge and restricted techniques make it necessary to design the knowledge base so that it can be easily altered and expanded by the end users.

Expert knowledge in the steel fabrication programme can be obtained from experienced steel fabrication estimators, detailers and engineers. These professionals possess information regarding the feasibility of different structural systems and the cost of producing them. Cooperation with industry and industrial partnership programmes between private corporations and the university play a key role in identifying and accessing such experts.

ii. Extracting Information

Once the available sources of information have been established, the appropriate information must be extracted from each source. The degree of processing required to extract useable information depends on the type of source and the information required.

Specific facts usually require little or no processing because the information is stated openly in the appropriate publications. In the case of steel fabrication, standards and section data are published in handbooks and industrial specifications.

Statistical information often requires the use of numerical techniques to extract useful data from the available records. Some information also requires synthetic adjustment and updating. These processes may require sophisticated statistical methods and powerful computers. The unit costs extracted from fabrication shop records are a prime example of this type of information [fig. 34a and 34b].

Expert knowledge is more difficult to extract. The mind of the expert can not always be accessed through direct questioning, nor can it be interfaced directly with powerful computer programmes. The decisions of a human expert are based on experience and intuition gained through education, creative thought and past mistakes. Personal preference and bias influence the decisions of many experts. In addition, an expert may not have isolated the particular basis for a decision and may be unable to explain it satisfactorily when questioned. The capture of expert knowledge [fig. 35] involves careful of questioning and observation of expert decisions followed by calibration and reevaluation of hypothesized knowledge. The process may help many experts uncover their personal biases and rationalize their thought processes.

Recall The Form Of The Estimating Equation:

$$\hat{c} = [q]\{\hat{u}\}$$

Using Observed Historical Data, This May Be Rewritten As:

$$\{\epsilon\} = \{C\} - \{\hat{C}\} = \{C\} - [Q]\{\hat{u}\}$$

Where:

$\{C\}$ is a column vector of observed project costs.

$\{\hat{C}\}$ is a column vector of calculated project costs.

$[Q]$ is an operations matrix in which each row is a project.

$\{\hat{u}\}$ is a column vector of calculated unit costs.

$\{\epsilon\}$ is a column vector of errors to be minimized.

Applying The Principle Of Least Squares And Minimizing $\{\epsilon\}$

$$\{\hat{u}\} = [[Q]^T[Q]]^{-1}[Q]^T\{C\}$$

For Convenience, This May Be Rewritten As:

$$\{\hat{u}\} = [A]\{C\}$$

In Summary:

$$\hat{c} = [q]\{\hat{u}\} = [q][A]\{C\} = [q][[Q]^T[Q]]^{-1}[Q]\{C\}$$

Figure 34a: Acquisition Of Mean Cost By Least Squares

Now The Variability Of The Prediction Can Be Established.

Calculate The Variation Of $\{\epsilon\}$

$$\hat{\sigma}^2 = \sum_{i=1}^{n} \frac{[C_i - [q_i]\{\hat{u}\}]^2}{n-(k+1)} = \frac{[\{C\} - [Q]\{\hat{u}\}]^T[\{C\} - [Q]\{\hat{u}\}]}{n-(k+1)}$$

Where:

$\hat{\sigma}^2$ is the variation of $\{\epsilon\}$

n is the number of rows of historical data.

k is the number of parameters in $\{\hat{u}\}$

As Observations Are Assumed To Be Independent Of One And Other, Define:

$$\sigma^2(\{C\}) = \hat{\sigma}^2[I]$$

Note That This Equation Can Be Transformed By $[A]$

$$\sigma^2([A]\{C\}) = [A]\sigma^2(\{C\})[A]^T$$

Therefore:

$$\sigma^2(\{\hat{u}\}) = [[Q]^T[Q]]^{-1}[Q]^T\sigma^2(\{C\})[Q][[Q]^T[Q]]^{-1}$$

Which Simplifies To:

$$\sigma^2(\{\hat{u}\}) = \hat{\sigma}^2[[Q]^T[Q]]^{-1}$$

Thus, The Variation Of A Predicted Project Cost Is:

$$\hat{\sigma}_C^2 = \hat{\sigma}^2[q]^T[[Q]^T[Q]]^{-1}[q]$$

Figure 34b: Acquisition Of Variation Of Costs By Least Squares

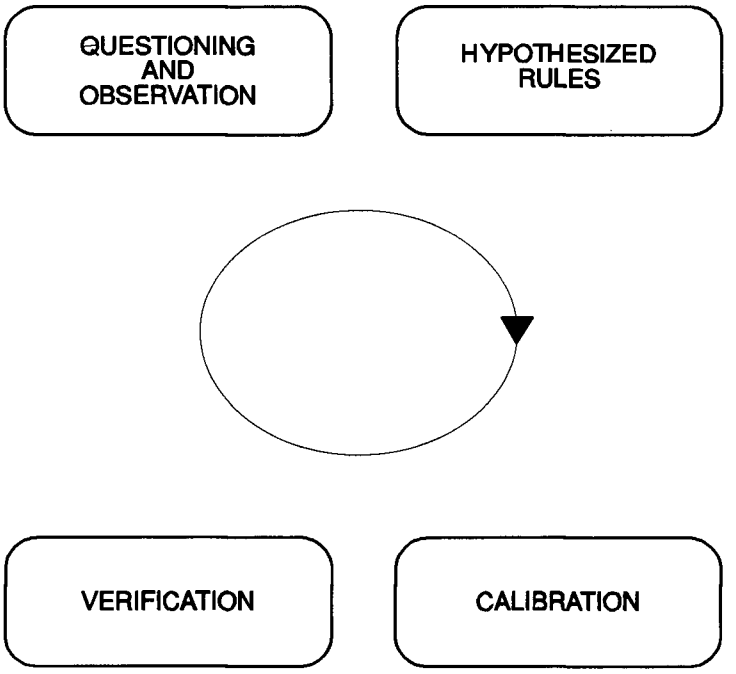


Figure 35: Capture Of Expert Knowledge

iii. Formatting And Storage

It is as important to store information properly as it is to locate and extract it from the original source. Improperly stored information is difficult to access and can impede the solution of a problem far more than any technical difficulty. Therefore, an efficient data format must be designed for the expert system to enhance its efficiency and encourage its use. The format must provide efficient computer access while remaining readable by the user. A black box approach should be avoided because it discourages the user from understanding and editing the knowledge base.

The proposed expert system stores data in text and spreadsheet files. Text files are used to store rules in a natural language format for the benefit of the end user. These rules are written using a rigid syntax that can be interpreted by a simple parser to produce a hierarchal tree of instructions for execution by the expert system. Other data, including the information stored in the working memory of the expert system, is kept in a spreadsheet format because spreadsheets are particularly well suited to processing tables of information. Most engineers are familiar with at least one spreadsheet environment.

Unfortunately, the spreadsheet format chosen for use in the expert system stored information sequentially and random access of the information stored on disk was not possible using low level functions. As a result, the operating speed of the system was significantly impaired. This is a classic example of selecting an inappropriate data format! It was initially selected because commercially available libraries promised easy integration of data with conventional C programmes and available spreadsheets.

iv. Processing Conflicting Strategies

As mentioned previously, expert decisions may be based on personal preference or bias. Experts do not always make the best decision. They may make mistakes, misunderstand the problem or overlook the best solution. They may even propose radically different solutions to the same problem. The developer of an expert system must select a strategy to handle such inconsistencies and conflicts.

The easiest strategy is to collect expert knowledge from a single source. This avoids the need to process conflicting information entirely; however, this approach produces an expert system that is a clone of a single expert incorporating all of his personal biases, preferences and inadequacies.

Another strategy is to collect information from a number of experts and to implement a composite method in the expert system which reflects the opinion of a majority of experts. This method is less likely to be influenced by the opinions of individual experts, so hopefully, individual biases are filtered out. Unfortunately, innovative solutions may also be eliminated. In addition, a broad information base minimizes the chance of overlooking promising solutions. This method will not necessarily guarantee an acceptable solution because the assumptions of various experts may be inconsistent.

The best strategy is to implement the techniques used by a number of experts in parallel. This approach is effectively a number of smaller expert systems linked together. It ensures that the assumptions and methods used are consistent and allows comparison of the solutions. In the end, the advice of a number of experts can be weighed before making a final selection. Severe disagreement in the conclusions of various experts may indicate the need for a more detailed investigation of the problem.

In the prototype expert system investigated, only one cost estimation method was implemented. It was not based the method of any established expert method because it was an attempt to incorporate new ideas into the field of cost estimation. Its development was justified as a tool to investigate the feasibility of new concepts. Ideally,

several methods should be implemented in parallel, but this is much more work. In any case, proper verification of the method is required before a commercial system can be developed.

v. Selecting Updating Criteria

The selection of updating criteria is another important aspect of information management. High quality information is required in order to produce high quality decisions and the quality of many types of information degrades with time. Appropriate criteria must be established to determine whether the available information is satisfactory or whether new information should be collected. The criteria must balance the cost of obtaining new information against the significance of changes in the data.

The criteria for updating information is heavily dependent on the type of information. Specific facts such as physical properties tend to be constant. Others such as industrial standards and section availability change slowly over time. Statistical information is often much more variable. The cost of operating a fabrication shop is constantly changing because it is a function of many internal and external factors which can not be completely controlled by the fabricator. Expert knowledge changes as new methods are developed.

C. Project Database

A project database [fig. 36] must be created for each project on the basis of a conventional quantity survey. It is easiest to prepare using a previous datafile as a template. It contains a detailed description of the type and size of each structural steel element. No updating of this database is required unless the proposed design of a structure is altered.

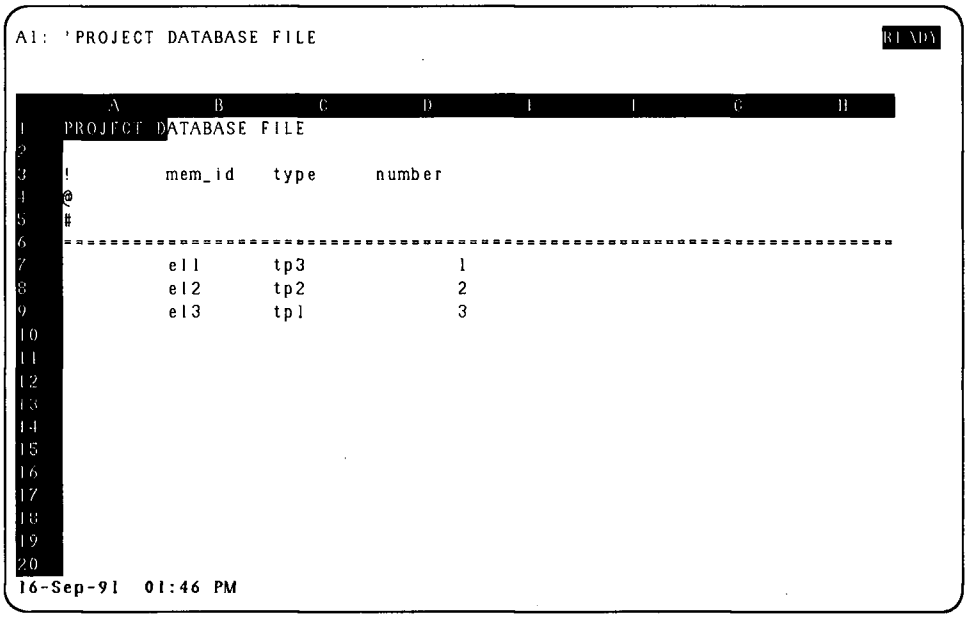


Figure 36: Project Database File Format

D. Operations Database

The operations database [fig. 37] is a permanent set of files containing the information necessary to convert the project database into a detailed description of the operations required to make each structural element. The database contains a set of standard members and connections which have been evaluated by experienced production experts. This set must be catalogued for reference by designers and fabricators. Elements which are not included in the database must be added before estimation can begin. This database must be updated and expanded as new standard members and connections are designed. It must also be updated when new production methods are introduced.

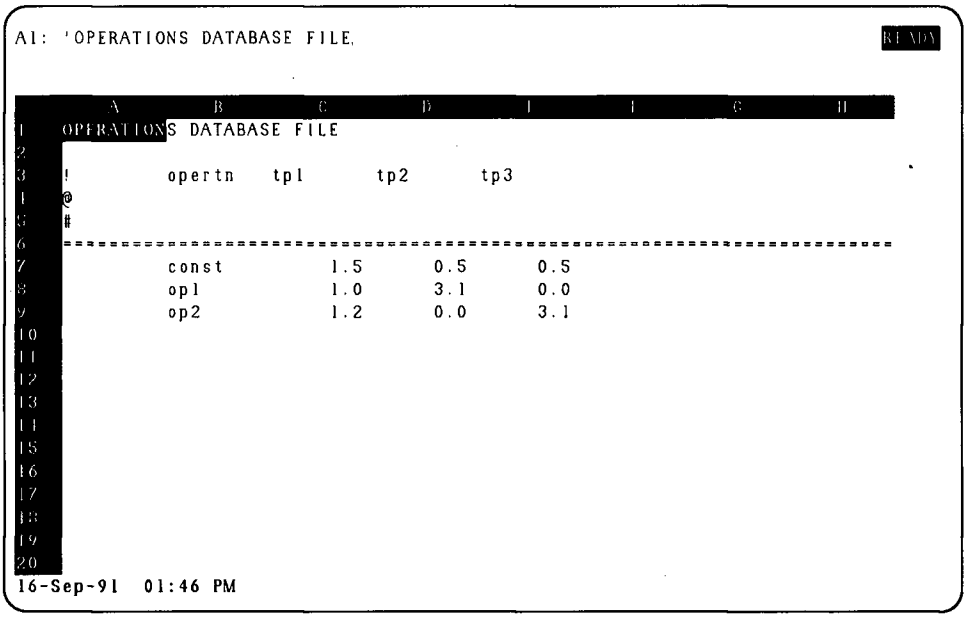


Figure 37: Operations Database File Format

E. Cost Database

The cost database [fig. 38] is a permanent database which reflects unit costs at a particular fabrication shop. It usually contains proprietary information which must be compiled directly from shop records using statistical methods. This information requires frequent updating to ensure that it represents current production costs. It must be altered whenever changes to the cost of material, labour or equipment are expected. In addition, a formalized feedback method must be developed to determine whether unit costs accurately reflect current information or whether they require updating.

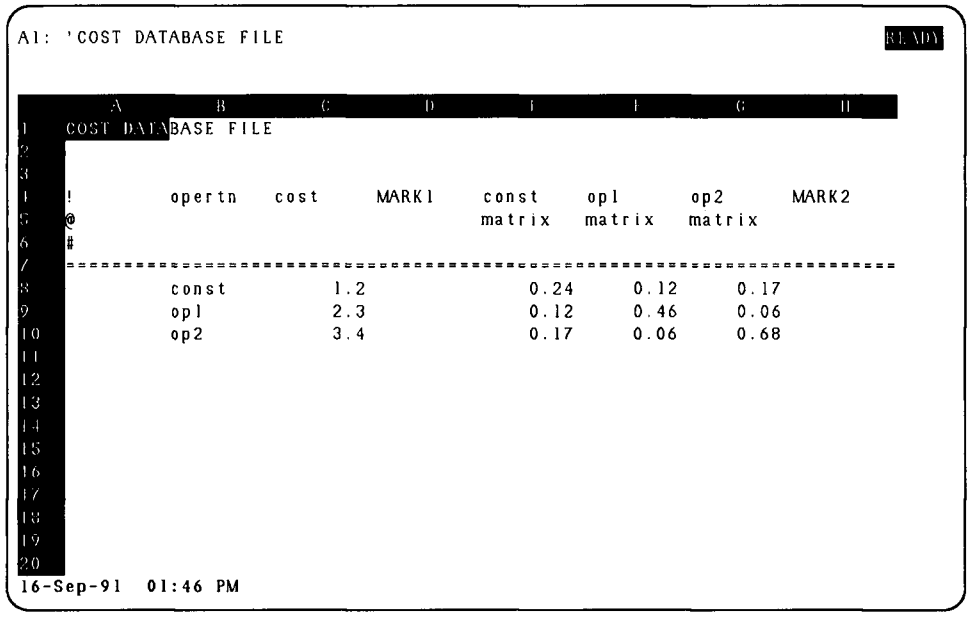


Figure 38: Cost Database File Format

F. Programme Output File

The programme output file is a permanent record of the estimate [fig. 39]. It summarizes the data used and shows all the calculations. It can be used to verify the results of the estimation process.

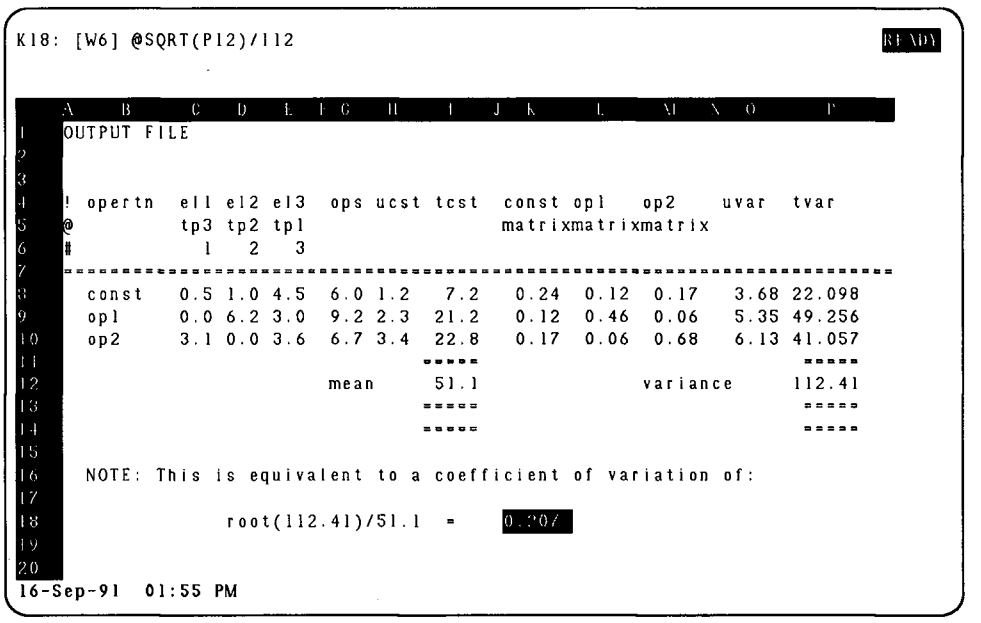


Figure 39: Output File Format

CHAPTER 10

AREAS FOR FURTHER RESEARCH

A. General Research

As with most research projects, this thesis has posed more new questions than it has answered. The basis of accepted cost estimation methods has been questioned and several new concepts have been proposed. Clearly, future research efforts will be required to resolve uncertainties and mature new concepts.

Careful consideration of the project has revealed five areas in which future research should be concentrated. Some of these areas reflect assumptions which were made to simplify analysis. Others represent topics which were apparent only after considerable effort had been committed to approaches which, in hindsight, were not the most efficient. These areas are:

a) accuracy of modelling;

b) the stochastic model;

c) accuracy of data collection;

d) programming techniques; and,

e) presentation and appearance.

B. Accuracy Of Modelling

The accuracy with which the cost model reflects reality is an important aspect of the cost estimation system because the results of the analysis can not be more accurate than the assumptions made in the cost model. For simplicity, the model was linear without interaction between operations. Clearly, this is not an accurate reflection of reality, but it is a convenient approximation.

Nonlinearities and interactions are an inherent part of the fabrication process. Interaction between operations is the result of bottleneck operations which disturb the flow of production and impair the efficiency of subsequent operations. Nonlinearities arise from the efficiency with which labour and equipment are utilized in each operation. Generally, the nonlinearities provide economies of scale which decrease the marginal cost of production for larger production volumes.

There are two promising means of improving the accuracy of the cost model which warrant further investigation:

a) the use of non-linear cost equations; and,
b) the use of direct simulation techniques.

Non-linear cost equations allow the cost of each operation to modelled more closely. A detailed study of each operation is required in order to justify a nonlinear cost equation. The equation may be derived analytically considering the subactivities of each operation or directly from historical data. In either case, this approach is penalized

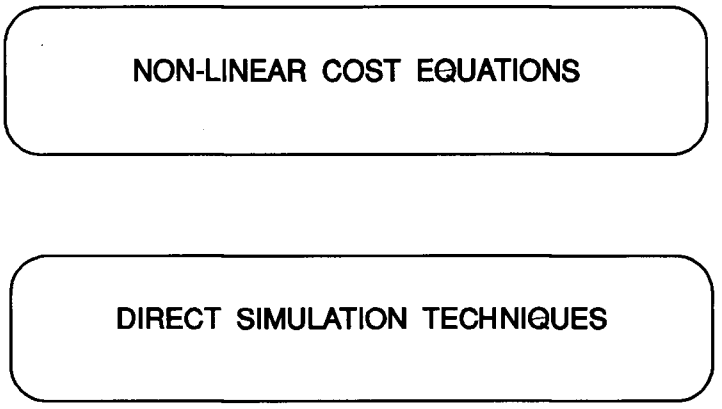


Figure 40: Means Of Improving Accuracy

because it requires a larger set of cost parameters which implies that a larger base of historical data is needed. In addition, a nonlinear model can not take advantage of the principle of superposition.

The use of direct simulation techniques provides a more accurate assessment of cost because the actual sequencing of fabrication operations is used to link subroutines emulating each fabrication operation. Nonlinearities and interaction are accounted for directly. Although this method requires the most effort, it is also the most rewarding because it can be used as an experimental tool to improve production and reduce cost.

C. The Stochastic Model

Stochastic modelling of cost is one of the new concepts introduced in this thesis. Although it is as yet unproven, it appears promising. This concept requires refinement and improvement before it becomes an accepted cost estimation tool.

The stochastic model described in this thesis is based on the Gaussian distribution. This distribution is a classic tool of statistical research. It is easy to work with and describes many physical situations; however, it may or may not provide the best description of fabrication cost. An argument against this distribution can be made because it allows cost values which are below zero and is not skewed. Real cost values must be greater than zero and are likely to be skewed because market conditions enforce a minimum cost but no maximum. Many other distributions exist which may be used to describe cost. In general, these distributions require more complex calculations than the Gaussian distribution, but these calculations are well within the capabilities of modern computer technology.

The application of the results of stochastic modelling is another area in which growth can be expected. While the stochastic model provides an indication of probable accuracy, it does not establish any criterion regarding the acceptable limits of accuracy. Such criterion must be developed in conjunction with industry. Also, a means of using feedback to verify the accuracy of the estimation system must be developed.

D. Accuracy Of Data Collection

The collection of accurate data is important in every method of cost estimation, but it is especially important when using the operations method. There are several aspects of this problem which deserve careful consideration:

a) access to information;
b) collection of historical information;
c) processing; and,
d) updating.

Access to information is a major problem to the university based investigator because cost data is proprietary information which can only be obtained through close liaison with industry. Although some contact with industry was made in the course of this thesis, no firm liaison was established. Future research would benefit from information that can only be obtained through an industrial partnership; however, such a liaison may reduce the freedom of the investigator to explore his own new concepts.

The collection of historical information depends upon shop records. The type and detail of these records determines the quality of information which can be derived. Although sophisticated shop floor information systems have been proposed since the 1960s, the steel fabrication industry appears to be slow to adopt such technology. Further research in the field of information management may result in improved information systems allowing cost data to be extracted more easily.

Processing of cost data from shop records is a computer-based activity requiring sophisticated statistical software. A detailed investigation of statistical methods may reveal improved methods of extracting data.

Accurate estimation requires current cost data. Some criterion must be established to determine the frequency with which data must be updated. Updating may involve the adjustment of existing information on the basis of feedback or it may require complete reprocessing of the cost database.

E. Programming Techniques

The expert system written for use in this thesis incorporates many features found in other expert systems with a few that appear to be unique. It is the only expert system encountered by the investigator that uses a spreadsheet environment for user interface. It is also the only one designed to perform its tasks on large sets of data. Refinement by an experienced programmer will improve the quality of the inference engine and its interface.

The choice of the C programming language was a sound decision because it is a powerful language that is commonly used; however, the recent introduction of object oriented C++ has provided a better choice. The expert system all ready makes extensive use of linked lists and pseudo-object oriented techniques. Rewritting the programme in C++ might result in faster more elegant code.

The spreadsheet data format is another feature of the programme which might be improved. Sequentially stored data inhibits the speedy retrieval of specific pieces of information. As a result, the expert system operates very slowly. This problem may be corrected by using a different data format. In addition, the WKS library used to access the spreadsheet data is lacking in some utilities. For example, the library contains utilities for writing to a spreadsheet but can not read the spreadsheet until it has been sorted. Unfortunately, no sorting utility is provided. This problem was overcome by writing a specialized sorting utility; nonetheless, a more complete library would be preferred.

F. Presentation And Appearance

Although the appearance of the expert system is adequate, it is not as sophisticated as commercially available shells. Careful presentation is important because it encourages use of the system and provides a tool for interpreting results. Aesthetically, the spreadsheet can be improved through the use of colours, windows and graphics. These improvements are beyond the scope of a research project but are important once the decision produce a commercial product is made. Improvements to presentation as a guide to interpretation are more important. The present user interface is capable of graphic presentation, but further work is required to determine what, if any, graphics are required to enhance interpretation.

CHAPTER 11

CONCLUSIONS

This thesis has examined the basis of current cost estimation techniques used in the steel fabrication industry. It has considered various aspects of the cost estimation problem and suggested improvements within the context of a fully automated expert system. It has attempted to strip away outmoded limitations and integrate innovative concepts to create a new cost estimation method that captures the subjective judgement of an experienced cost estimator within a detailed knowledge base. In addition, it has experimented with the development of an expert system specifically for use in engineering problems which involve large amounts of data and complex calculations. The key points of this development are reiterated here to emphasize the more important findings of this research.

The development of new tools for cost estimation is overwhelmingly justified by economic considerations because cost information is the basis of many critical decisions in the steel construction industries. Such tools provide a means of improving the accuracy of cost estimation so that reliable economic decisions can be made, thereby ensuring a healthy and competitive steel construction industry.

Cost estimation aids magnify the effect of direct human effort providing leverage which makes detailed estimation economically feasible. Computers are the penultimate estimation aid as they are capable of replacing all the components of cost estimation including the subjective judgement of the estimator himself. Computer technology has been accepted by industry with the advent of powerful, affordable personal computers.

They are capable of rapid, accurate data manipulation and calculation which may be used to overcome the practical constraints that have limited the use of extensive and complex calculations in traditional estimation methods.

The introduction of computer technology into the office environment provides an opportunity to introduce new concepts and techniques into the field of cost estimation. It is important to recognize that direct analogs of traditional estimation methods may not be optimal. New methods must be developed specifically for a computer-based environment. These methods must exploit the strengths and avoid the weakness of computer technology.

The replacement of human estimation expertise by a computer-based expert systems has been demonstrated in principle, although future development will be required before such systems can be introduced commercially. Expert systems reduce the reliance of industry on human experts by harnessing the full power of computer technology. They improve access to expert knowledge by providing expertise in a packaged form which is always available yet may be stored indefinitely. In addition, they can be quickly duplicated or transmitted electronically as required. Moreover, they execute their tasks in a consistent unbiased manner making them ideal for the comparison of alternates.

The development of a new cost estimation method for implementation in an expert system has allowed the full power of computer technology to be focused on the cost estimation problem. While traditional cost estimation methods are limited by constraints which are not valid in a computer environment, the new method is free to

exploit the data manipulation and reliable calculation abilities of computer technology. It is designed to minimize direct human interface in the estimation process and is not dependent on subjective human thought. Communication between the user and the expert system is streamlined through the use of standardization which increases speed and minimizes the influence of the operator on the final results ensuring that the estimate is unbiased. More importantly, it enables estimation to be performed by a non-expert.

The new method is based on the operations approach to cost estimation which is the most promising because it incorporates project specific information regarding the materials, labour and equipment required for fabrication. It uses a high level of detail allowing it to provide an accurate prediction of cost; however, it requires a prohibitive amount of effort when manual methods of estimation are used. It is ideal for use in an expert system environment because it uses detailed calculation to replace subjective judgement found in less detailed methods of cost estimation. The need for subjective judgement can be minimized if only physical parameters are chosen for the cost model because they can be obtained directly from a quantity survey without expert knowledge. Physical parameters provide a good indicator of direct cost. With appropriate assumptions, they can also be used to assess indirect costs, thereby minimizing the total number of parameters.

It is important that the expert system be easy to learn and use. Object-oriented techniques and commonly accepted utilities were incorporated into the expert system shell to ensure that it is portable and user friendly. The system can be easily customized to use other text and spreadsheet editors to provide a familiar working environment

and minimize the programming burden. It is written in the C programming language which is a powerful and accepted language. In addition, it uses linked lists and branched linked lists to make optimum use of the available memory. Unlike other expert systems, it is intended to process the mathematical operations frequently used by engineers.

Knowledge of the variation of costs is as important as knowledge of the mean for the comparison of estimates and the preparation of bids. The use of stochastic methods is an innovation which allows detailed information on the distribution of cost to be estimated. This information is invaluable because it enables the quality of an estimate to be explicitly calculated in terms of accuracy. It can also be used in the comparison of alternates.

Knowledge acquisition and database maintenance are vital tasks in the development of an expert system. The knowledge base must be flexible so that it can be altered to include new information. Flexibility is important because much of the necessary information for cost estimation is proprietary in nature and can not be accessed by the developer of the expert system. Expert knowledge must be painstakingly isolated and incorporated into the expert system in the form of rules. Statistical information, such as the unit costs of production, must be derived from historical records. All information in the knowledge base must be updated periodically to ensure that it reflects current data and practices. Cost data in particular requires careful monitoring.

BIBLIOGRAPHY

1. Adeli, H.; Balasubramanyag, *A Knowledge-Based System For Design Of Bridge Trusses.* ASCE Journal Of Structural Engineering, Vol. 2, No. 1, 1-20, 1988.

2. Ahmad, Irtishad; Minkarah, Issam, *An Expert System For Selecting Bid Markups.* Computers In Civil Engineering, 229-237, 1988.

3. *Artificial Intelligence In Manufacturing.* Elsevier Science Publishers, Amsterdam, 1985.

4. Austin, M.A.; Mahin, S.A.; Pister, K.S. *CSTRUCT: Computer Environment For Design Of Steel Structures.* ASCE Journal Of Structural Engineering, Vol. 3, No. 3, 209-225, 1989.

5. Berry, G.L. *Shop Floor Information System: Design And Implementation.* Engineering Digest, 1984.

6. Bradford, Henry, *The Dozen Best Roadblocks To Automation.* Computers In Civil Engineering, 802-805, 1988.

7. Bristol, Charles R.; Marks, Raymond A.; Costea, Jill A. *Computerized Cost Estimation In County Government.* Computers In Civil Engineering, 477-490, 1986.

8. Chang, T.C.; Wysk, Richard A. *An Introduction To Automated Process Planning Systems.* Prentice Hall, Englewood Cliffs, New Jersey, 1985.

9. Chiang, Kai; Gergely, Peter, *Interactive Structural Analysis And Steel Design On The Macintosh.* Electronic Computation, 570-578, 1986.

10. Choudhary, Kahlid Tanwir, *Cost Estimation Of Industrial Buildings.* Thesis, Concordia University, 1978.

11. Cronembold, J.R.; Law, Kincho H. *Automated Processing Of Design Standards.* ASCE Journal Of Structural Engineering, Vol. 2, No. 3, 255-273, 1988.

12. Devore, Jay L. *Probability And Statistics For Engineering And The Sciences.* Brooks/Cole Publishing Company, Monterey, California, 1982.

13. Fenves, Gregory L. *Object Representations For Structural Analysis And Design.* Computers In Civil Engineering, 502-511, 1988.

14. Fenves, S.J.; Flemming, V.; Hendrickson, C.; Maher, M.L.; Schmitt, G. *An Integrated Software Environment For Building Design And Construction.* Computers In Civil Engineering, 21-32, 1988.

15. Fenves, S.J.; Maher, M.L.; Sriram, D. *Knowledge-Based Expert Systems In Civil Engineering.* Computers In Civil Engineering, 248-257, 1984.

16. Finn, Gavin A.; Reinschmidt, Kenneth F. *Microcomputer-Based Engineering Expert Systems.* Computers In Civil Engineering, 812-826, 1986.

17. Flachsbart, Barry B. *Reflections On The Impact Of Computer Science On Engineering.* Electronic Computation, 14-20, 1986.

18. Forde, Bruce W.R. *An Application Of Selected Artificial Intelligence Techniques To Engineering Analysis.* Thesis, University Of British Columbia, 1989.

19. Gifford, J.B. *Microcomputers In Civil Engineering: Use And Misuse.* ASCE Journal Of Structural Engineering, Vol. 1, No. 1, 61-68, 1987.

20. Kernighan, Brian W.; Ritchie, Dennis, M. *The C Programming Language.* Prentice Hall, Englewood Cliffs, New Jersey, 1988.

21. Law, Kincho H.; Jouanem, Mazen K. *Data Modelling For Building Design.* Computers In Civil Engineering, 21-36, 1986.

22. Leung, Y.C. *A Contribution To Computer Aided Design Evaluation Of Steel Structures.* Thesis, University of British Columbia, 1984.

23. Maher, M.L. *Expert Systems For Structural Design.* ASCE Journal Of Structural Engineering, Vol. 1, No. 4, October, 1987.

24. Milner, D.A.; Vasiliou, V.C. *Computer Aided Engineering For Manufacture.* Kogan Page, London, England, 1986.

25. Naeim, Farzad; Dehghanyar, T.J. *Building Design Language.* Computers In Civil Engineering, 573-581, 1988.

26. Naeim, Farzad; Martin, John A. *Applications Of Artificial Intelligence In Preliminary Structural Design.* Electronic Computation, 53-64, 1986.

27. Navin, F.P.D.; Stiemer, S.F. *Engineering With Spreadsheets: An Electronic Textbook.* Maplesoft, Vancouver, 1991.

28. Negoita, Constantin Virgil, *Expert Systems And Fuzzy Systems.* Benjamin/Cummins Publishing, 1985.

29. Nixon, D. *Estimating The Cost Of Small Steel Buildings.* Canadian Journal Of Civil Engineering, Vol. 1, No. 2, 1974.

30. O'Connor, M.J.; Lemberger, Ellen S., *The Use Of Focus Database Management Systems In A Cost Estimating System.* Computers In Civil Engineering, 564-477, 1978.

31. Orenstein, Glenn S. *Instant Expertise: A Danger Of Small Computers.* Computers In Civil Engineering, 578-582, 1984.

32. Paek, Y.J.; Adeli, H. *Representation Of Structural Design Knowledge In A Symbolic Language.* ASCE Journal Of Structural Engineering, Vol. 2, No. 4, 346-363, 1988.

33. Parsave; Chignell; Khoshafian; Wong, *Intelligent Data Bases.* John Wiley And Sons, New York, 1989.

34. Pixley, Ray A.; Ridlon, Stephen, *How To Check An Engineering Computer Program.* Computers In Civil Engineering, 583-593, 1984.

35. Plotnick, Paul H. *Computer Impact Of Construction.* Computers In Civil Engineering, 390-396, 1978.

36. Prisig, Robert M. *Zen And The Art Of Motorcycle Maintenance: An Inquiry Into Values.* Bantam Books, Toronto, 1988.

37. Rasdorf, William J,; Parks, Linda M. *Expert Systems And Engineering Design Knowledge.* Electronic Computation, 28-42, 1986.

38. Rasdorf, William J.; Wang, TsoJen, *Expert System Integrity Maintenance For The Retrieval Of Data From Engineering Databases.* Computers In Civil Engineering, 654-668, 1986.

39. Russell, A.D. *Cost Optimization Of A structural Roof System.* Thesis, University Of British Columbia, 1969.

40. Salazar, Guillermo F. *Microcomputer Tunneling Cost Estimation.* Computers In Civil Engineering, 461-470, 1988.

41. Sause, Richard; Powell, Graham, *Knowledge Representation And Processing For Computer Integrated Structural Design.* Computers In Civil Engineering, 1-10, 1988.

42. Schaefer, Robert S.; Tundermann, Stephen M.; Pesqera, Carlos I.; Abel, John F. *Experience With Workstation Based Design Of Steel Structures.* Electronic Computation, 374-382, 1986.

43. Schildt, Herbert. *Artificial Intelligence Using C.*, McGraw-Hill, Berkeley, California, 1987.

44. Shing, W.Y. Albert, *Computer Aided Cost Estimation Of Steel Structures: A Case Study Of Operational Approach.* Thesis, University Of British Columbia, 1986.

45. Soh, Chee-Koing; Soh, Ai-Kah, *Example Of Intelligent Structural Design System.* ASCE Journal Of Structural Engineering, Vol. 2, No. 4, 329-345, 1988.

46. Stroustrup, Bjarne *The C++ Programming Language.* Addison Wesley Publishing, Don Mills, Ontario, 1987.

47. Stahl, Fred I. *The Standards Interface For Computer Aided Design: An Overview Of Some Technical Problems Associated With Automated Design Checking.* Computers In Civil Engineering, 560- 567, 1984.

48. Tocher, James L. *A Perspective On Engineering Computing.* Electronic Computation, 21-27, 1986.

49. Trefzer, Felix, *Standard Costing: Basis Of Project Management.* Computers In Civil Engineering, 604-618, 1981.

50. Wentorf, R.; Cronembold, Jose R.; Law, Kincho H. *Integration Of Modelling, Analysis And Design.* Computers In Civil Engineering, 134-143, 1988.

APPENDIX A
LEAST SQUARES: AN EXAMPLE

This appendix contains a detailed description of the method of least squares which has been proposed as a method for determining the unit cost vector and its corresponding covariance matrix.

FROM PREVIOUS ASSUMPTIONS:

$$\{C\} = [Q]\{u\}$$

WHERE:

$\{C\}$ is an exact project cost vector.
$[Q]$ is a matrix of project operations.
$\{u\}$ is an exact unit cost vector.

HOWEVER, IT IS UNLIKELY THAT EITHER VECTOR IS EXACT, SO A BEST FIT IS SOUGHT BY MINIMIZING THE ERROR:

$$\{\epsilon\} = \{C\} - \{\hat{C}\} = \{C\} - [Q]\{\hat{u}\}$$

WHERE:

$\{\epsilon\}$ is an error vector.
$\{\hat{u}\}$ is an estimated unit cost vector.

NOTE THAT THIS IMPLIES:

$$\{\epsilon\}^T = \{C\}^T - \{\hat{u}\}^T[Q]^T$$

NOW, THE SUM OF THE SQUARES OF THE ERROR TERM MAY BE EXPRESSED AS A SCALAR:

$$\{\epsilon\}^T\{\epsilon\} = (\{C\}^T - \{\hat{u}\}^T[Q]^T)(\{C\} - [Q]\{\hat{u}\})$$

$$\{\epsilon\}^T\{\epsilon\} = \{C\}^T\{C\} - \{C\}^T[Q]\{\hat{u}\} - \{\hat{u}\}^T[Q]^T\{C\} + \{\hat{u}\}^T[Q]^T[Q]\{\hat{u}\}$$

$$\{\epsilon\}^T\{\epsilon\} = \{C\}^T\{C\} - 2\{\hat{u}\}^T[Q]^T\{C\} + \{\hat{u}\}^T[Q]^T[Q]\{\hat{u}\}$$

THE OBJECT OF LEAST SQUARES IS TO MINIMIZE THIS SCALAR, SO THE DERIVATIVE OF THE FUNCTION IS SET TO ZERO:

$$0 = -2[Q]^T\{C\} + 2[Q]^T[Q]\{\hat{u}\}$$

SIMPLIFYING THE ABOVE EXPRESSION:

$$\{\hat{u}\} = [[Q]^T[Q]]^{-1}[Q]^T\{C\}$$

A SIMILAR APPROACH CAN BE USED TO VERIFY THE ESTIMATOR USED TO PREDICT THE COVARIANCE MATRIX.

APPENDIX B

DIRECTORY STRUCTURE AND PROGRAMMES

ROOT DIRECTORY		
AUTOEXEC.BAT		
CNTRL DIRECTORY	**SHOP DIRECTORY**	**JOB DIRECTORY**
C_RULEDT.EXE C_DATEDT.EXE C_CONFIG.EXE C_CNTRL.EXE C_SHOP.EXE C_NGIN.EXE C_JOB.EXE	OPERTN.WK1 COST.WK1 TO.WK1	JOBLIST.WK1 OPERTN.WK1 WORK.WK1 COST.WK1 TO.WK1
C_CONFIG.DAT RULE.DAT	C_BATCH.DAT	C_BATCH.DAT
VIP_CALC.EXE NE.COM	NE.COM	NE.COM
C_BATCH.BAT C_JOB.BAT		

APPENDIX C
PROGRAMME SOURCE CODE

C_CNTRL.PRJ

```
c_cntrl
c_config
```

C_CNTRL.H

```
/*********************************************************/
/*                                                       */
/*   This is the header file for c_cntrl.c               */
/*                                                       */
/*********************************************************/

#include <stdio.h>
#include <process.h>
#include <conio.h>
#include <ctype.h>
#include <string.h>
#include <alloc.h>

#include "c_config.h"
```

C_CNTRL.C

```
/*********************************************************/
/*                                                       */
/*   This program acts as a controller for the expert    */
/*   expert system. It contains a menu which provides    */
/*   access to the various editors and utilities of the  */
/*   expert system.                                      */
/*                                                       */
/*********************************************************/

#include "c_cntrl.h"
```

```
main()
{
   int status;
   char ch;
   char *path=NULL;
   char path_cntrl[80];
   char *args[1];
   char *message = "Fatal Error";


   do {
       printf("\n\n");
       printf("CXS EXPERT SYSTEM \n");
       printf("Menu:  C - Edit Configuration File \n");
       printf("       J - Prepare Job Directory \n");
       printf("       S - Prepare Shop Directory \n");
       printf("       R - Edit Rule Base On Configuration \n");
       printf("       B - Edit Data Base On Configuration \n");
       printf("       X - Execute Rule Base On Configuration \n");
       printf("       Q - Quit Program \n");
       printf("\nSelection : ");
       ch=toupper(getche());
       printf("\n\n");

       path=strcpy(path_cntrl,c_config("CNTRL"));

       switch(ch)  {
            case 'C': strcat(path,"C_CONEDT.EXE");
                      args[0]="C_CONEDT.EXE";
                      status=spawnl(P_WAIT,path,args[0],NULL);
                      if(status==-1) perror(message);
                      break;
            case 'J': strcat(path,"C_JOB.EXE");
                      args[0]="C_JOB.EXE";
                      status=spawnl(P_WAIT,path,args[0],NULL);
                      if(status==-1) perror(message);
                      break;
            case 'S': strcat(path,"C_SHOP.EXE");
                      args[0]="C_SHOP.EXE";
                      status=spawnl(P_WAIT,path,args[0],NULL);
                      if(status==-1) perror(message);
                      break;
            case 'R': strcat(path,"C_RULEDT.EXE");
                      args[0]="C_RULEDT.EXE";
                      status=spawnl(P_WAIT,path,args[0],NULL);
                      if(status==-1) perror(message);
                      break;
            case 'B': strcat(path,"C_DATEDT.EXE");
                      args[0]="C_DATEDT.EXE";
                      status=spawnl(P_WAIT,path,args[0],NULL);
                      if(status==-1) perror(message);
                      break;
            case 'X': strcat(path,"C_NGIN.EXE");
                      args[0]="C_NGIN.EXE";
                      status=spawnl(P_WAIT,path,args[0],NULL);
                      if(status==-1) perror(message);
                      break;
       }
```

```
   } while(ch!='Q');
}
```

C_CONFIG.H

```
/*************************************************************/
/*                                                           */
/*   This is the header file for c_config.c                  */
/*                                                           */
/*************************************************************/

# include <stdio.h>
#include <string.h>

char* c_config( char *);
```

C_CONFIG.C

```
/*************************************************************/
/*                                                           */
/*   This is the header file for c_config.c                  */
/*                                                           */
/*************************************************************/

# include <stdio.h>
#include <string.h>

char* c_config( char *);
```

C_CONFIG.DAT

```
CNTRL C:\TC\
SHOP C:\TC\SHOP\
JOB C:\TC\JOB\
```

C_CONEDT.PRJ

```
c_conedt
c_config
```

C_CONEDT.H

```
/*********************************************************/
/*                                                       */
/*   This is the header file for c_conedt.c              */
/*                                                       */
/*********************************************************/

#include <stdio.h>
#include <process.h>
#include <string.h>

#include "c_config.h"
```

C_CONEDT.C

```
/*********************************************************************/
/*                                                                   */
/*   This programme edits the configuration file from within         */
/*   expert system environment. Note that although the Norton        */
/*   Editor NE.COM is used, the programme can be easily              */
/*   customized.                                                     */
/*                                                                   */
/*********************************************************************/

#include "c_conedt.h"

main()
{
   int status;
   char path_cntrl[80];
   char *path;
   char *args[2];
   char *message = "Fatal Error";


   path=strcpy(path_cntrl,c_config("CNTRL"));

   args[0]="NE.COM";
   path=strcat(path,args[0]);

   args[1]="C_CONFIG.DAT";
   args[2]=NULL;
```

```
   status=spawnv(P_WAIT,path,args);
   if(status==-1) perror(message);
}
```

C_JOB.PRJ

```
c_job
c_config
```

C_JOB.H

```
/*********************************************************/
/*                                                       */
/*   This is the header file for c_job.c                 */
/*                                                       */
/*********************************************************/

#include <stdio.h>
#include <ctype.h>
#include <conio.h>
#include <stdlib.h>

#include "c_config.h"
```

C_JOB.C

```
/*********************************************************************/
/*                                                                   */
/*  This programme uses a DOS batch file to prepare a directory      */
/*  containing all the data necessary for each job. This avoids      */
/*  when background information is updated.                          *
/*                                                                   */
/*********************************************************************/

#include "c_job.h"

void main()
{
   int i,ch;
   char cntrl[80];
   char source[80];
   char destination[80];
   char command[80];
```

```
    strcpy(cntrl,c_config("CNTRL"));
    i=strlen(cntrl);
    cntrl[i-1]='\0';
    printf("CNTRL is %s \n",cntrl);

    strcpy(source,c_config("SHOP"));
    i=strlen(source);
    source[i-1]='\0';
    printf("SHOP is %s \n",source);

    strcpy(destination,c_config("JOB"));
    i=strlen(destination);
    destination[i-1]='\0';
    printf("JOB is %s \n",destination);

    printf("Confirmation (Y/N)");
    for(;;) {
        ch=toupper(getch());
        if(ch=='N') return;
        if(ch=='Y') break;
    }
    printf("\n");

    sprintf(command,"c_batch ");
    strcat(command,cntrl);
    strcat(command," ");
    strcat(command,source);
    strcat(command," ");
    strcat(command,destination);

    system(command);
}
```

C_BATCH.BAT

```
echo root %1
echo source %2
echo destination %3
pause
if exist %3\c_batch.dat goto continue
md %3
pause
:continue
copy %2\*.* %3
cd %3
pause
ne c_batch.dat
cd %1
```

C_SHOP.PRJ

```
c_shop
c_config
```

C_SHOP.H

```
/*********************************************************/
/*                                                       */
/*   This is the header file for c_shop.c                */
/*                                                       */
/*********************************************************/

#include <stdio.h>
#include <process.h>
#include <string.h>
#include <conio.h>
#include <ctype.h>

#include "c_config.h"
```

C_SHOP.C

```
/*********************************************************/
/*                                                       */
/*   This programme provides access to the shop_file     */
/*   spreadsheet editor, in this case, VIP_CALC. Again,  */
/*   the spreadsheet editor can be easily customized.    */
/*                                                       */
/*********************************************************/

#include "c_shop.h"

main()
{
   int status;
   char input[30];
   char path_shop[80];
   char path_cntrl[80];
   char *args[2];
   char *message = "Fatal Error";

   args[0]="VIP_CALC.EXE";
   strcpy(path_cntrl,c_config("CNTRL"));
   strcat(path_cntrl,args[0]);

   strcpy(path_shop,c_config("SHOP"));
```

```
   printf("Edit File : ");
   gets(input);
   args[1]=strcat(path_shop,input);
   printf("\n");
   args[2]=NULL;

   status=spawnv(P_WAIT,path_cntrl,args);
   if(status==-1) perror(message);

}
```

C_RULEDT.PRJ

```
c_ruledt
c_config
```

C_RULEDT.H

```
/*********************************************************/
/*                                                       */
/*   This is the header file to c_ruledt.c               */
/*                                                       */
/*********************************************************/

#include <stdio.h>
#include <process.h>
#include <string.h>

#include "c_config.h"
```

C_RULEDT.C

```
/*********************************************************/
/*                                                       */
/*   This program edits the rule base which is stored    */
/*   in ASCII and accessed by the inference engine. It   */
/*   uses the Norton Editor NE.COM                       */
/*                                                       */
/*********************************************************/

#include "c_ruledt.h"
```

```
main()
{
   int status;
   char path_cntrl[80];
   char *path;
   char *args[2];
   char *message = "Fatal Error";


   path=strcpy(path_cntrl,c_config("CNTRL"));

   args[0]="NE.COM";
   path=strcat(path,args[0]);

   args[1]="RULE.DAT";
   args[2]=NULL;

   status=spawnv(P_WAIT,path,args);
   if(status==-1) perror(message);

}
```

C_DATEDT.PRJ

```
c_datedt
c_config
```

C_DATEDT.H

```
/*********************************************************/
/*                                                       */
/*   This is the header file for c_datedt.c              */
/*                                                       */
/*********************************************************/

#include <stdio.h>
#include <process.h>
#include <string.h>
#include <conio.h>
#include <ctype.h>

#include "c_config.h"
```

C_DATEDT.C

```
/*********************************************************/
/*                                                       */
/*   This program edits background data in the job and   */
/*   shop directories. This data is stored in a          */
/*   spreadsheet format and is accessed by the VIP_      */
/*   CALC editor.                                        */
/*                                                       */
/*********************************************************/

#include "c_datedt.h"

main()
{
   char ch;
   int status;
   char input[30];
   char path_dat[80];
   char path_cntrl[80];
   char *args[2];
   char *message = "Fatal Error";

   args[0]="VIP_CALC.EXE";
   strcpy(path_cntrl,c_config("CNTRL"));
   strcat(path_cntrl,args[0]);

   for(;;) {
       printf("Is it a shop file or a job file (S/J) :");
       ch=toupper(getch());
       printf("\n");

       if(ch=='S' || ch=='J') break;
   }

   switch(ch) {
       case 'S':strcpy(path_dat,c_config("SHOP"));
            break;
       case 'J':strcpy(path_dat,c_config("JOB"));
            break;
   }

   printf("Edit File : ");
   gets(input);
   args[1]=strcat(path_dat,input);
   printf("\n");
   args[2]=NULL;

   status=spawnv(P_WAIT,path_cntrl,args);
   if(status==-1) perror(message);

}
```

C_NGIN.PRJ

```
c_ngin
n_call
n_caif
n_file
n_stck
c:\tc\lib\lwks.lib
c_config
```

C_NGIN.H

```
/*********************************************************/
/*                                                       */
/*   This is the header file for c_ngin.c                */
/*                                                       */
/*********************************************************/

#include <stdio.h>
#include <alloc.h>
#include <string.h>
#include <conio.h>

#include "n_call.h"
#include "n_caif.h"

#include "n_file.h"
#include "n_stck.h"

#include "c_config.h"

int c_ngin(void);
int parse(char*, char*, char*);
int call(char*);
int callif(char*);
```

C_NGIN.C

```
/*********************************************************/
/*                                                       */
/*   This programme is the main body of the inference    */
/*   engine. It interprets the rule base and executes    */
/*   subsidiary modules designated n_****.               */
/*                                                       */
/*********************************************************/

#include "c_ngin.h"
```

```
struct unit {
   char name[80];
   struct unit *next;
   struct unit *branch;
};

struct stack {
   struct unit *stack;
   struct stack *prev;
};

struct unit *start_rule;
struct unit *start_data;

/* Main module of the inference engine */

void main()
{
   int status;
   status=c_ngin();
   if(status==0) printf("\n\n Error In NGIN\n\n ");
   if(status==1) printf("\n\n NGIN Sucessful - Rule Base Processed \n\n");
}

/* Inference engine module accessing rule base */

 int c_ngin()
{
   int spc;
   char frule[80];
   char cline[80], command[80], predicate[80];
   struct unit *u, *v;
   struct stack *s;

   strcpy(frule,c_config("CNTRL"));
   strcat(frule,"RULE.DAT");

   spc=0;

   s=(struct stack *) new_stack();
   if(!s) return 0;

   start_rule=(struct unit *) rule_load(frule);
   if(start_rule==0) return 0;
   u=start_rule;
   u=u->next;
   if(!u) return 0;

   for(;;) {
      strset(cline,' ');
      strset(command,' ');
      strset(predicate,' ');

      strcpy(cline,u->name);

       if(parse(cline,command,predicate)==0) return 0;

       if(strcmpi(command,"IF")==0 && strcmpi(command,"IFNOT")==0 && strcmpi(command,"CALL")==0 &&
strcmpi(command,"REM")==0) return 0;
```

```
if(strcmpi(command,"IF")==0) {
      int truth;
      truth=callif(predicate);
      if(truth==0) return 0;
      if(truth==-1) {
           v=u->next;
           if(v) u=v;
           if(!v) {
              for(;;) {
                   spc--;

                   s=(struct stack *) pull_off(s);
                   if(!s) return 1;
                   v=s->stack;
                   v=v->next;
                   if(v) break;
              }
              u=v;
           }
      }
      if(truth==1) {
           v=u->branch;
           if(!v) {
              printf("Error In Syntax\n");
              return 0;
           }

           s=(struct stack *) push_on(s,u);
           if(!s) return 0;

           if(v) u=v;
      }
}

 if(strcmpi(command,"IFNOT")==0) {
      int truth;
      truth=callif(predicate);
      if(truth==0) return 0;
      if(truth==1) {
           v=u->next;
           if(v) u=v;
           if(!v) {
             for(;;) {
                   spc--;

                   s=(struct stack *) pull_off(s);
                   if(!s) {
                        printf("\n\n Rule - Base Processed \n\n");
                        return 1;
                   }
                   v=s->stack;
                   v=v->next;
                   if(v) break;
              }
              u=v;
           }
      }
      if(truth==-1) {
           v=u->branch;
              if(!v) {
```

```
                    printf("Error In Syntax\n");
                    return 0;
                }

                s=(struct stack *) push_on(s,u);
                if(!s) return 0;

                if(v) u=v;
            }
        }

        if(strcmpi(command,"CALL")==0) {
            if(call(predicate)==0) return 0;
            v=u->next;
            if(!v) {
                s=(struct stack *) pull_off(s);
                if(!s) {
                    printf("\n\n Rule - Base Processed \n\n");
                    return 1;
                }
                v=s->stack;
                v=v->next;
                if(!v) {
                    printf("\n\n Rule - Base Processed \n\n");
                    return 1;
                }
            }
            u=v;
        }

        if(strcmpi(command,"REM")==0) {
            v=u->next;
            if(!v) {
                s=(struct stack *) pull_off(s);
                if(!s) {
                    printf("\n\n Rule - Base Processed \n\n");
                    return 1;
                }
                v=s->stack;
                v=v->next;
                if(!v) {
                    printf("\n\n Rule - Base Processed \n\n");
                    return 1;
                }
            }
            u=v;
        }
    }
}

/* Parser to interpret rules and execute subsidiary modules */

int parse(char *cline, char* command, char* predicate)
{
    char ch;
    int i,j;
    for(i=0;i<80;i++) {
        ch=cline[i];
        if(ch==' ') {
        command[i]='\0';
```

```
      break;
      }
      command[i]=cline[i];
   }
   for(j=i+1;j<80;j++) predicate[j-(i+1)]=cline[j];
   return 1;
}

/* Control module for direct CALL commands */

int call(char* cline)
{
   char path_job[80];
   char command[80], predicate[80];
   char file1[80], head1[80], file2[80], head2[80], head3[80], head4[80];
   char key_col[80], key_col2[80], file3[80];
   char operator[80];
   int i;

   if(parse(cline,command,predicate)==0) return 0;

   if(strcmpi(command,"comment")==0) {
      i=call_comment(predicate);
      return i;
   }

   if(strcmpi(command,"message")==0) {
       i=call_message(predicate);
       return i;
   }

   if(strcmpi(command,"copy_file")==0) {
       if(parse(predicate,file1,file2)==0) return 0;
       strcpy(file1,strcat(strcpy(path_job,c_config("JOB")),file1));
       strcpy(file2,strcat(strcpy(path_job,c_config("JOB")),file2));
       i=call_copy_file(file1, file2);
       return i;
   }

   if(strcmpi(command,"edit_file")==0) {
       strcpy(predicate,strcat(strcpy(path_job,c_config("JOB")),predicate));
       i=call_edit_file(predicate);
       return i;
   }

   if(strcmpi(command,"input_heading")==0) {
       if(parse(predicate,file1,predicate)==0) return 0;
       strcpy(file1,strcat(strcpy(path_job,c_config("JOB")),file1));
       if(parse(predicate,head1,predicate)==0) return 0;
       if(parse(predicate,head2,head3)==0) return 0;
       i=call_input_heading(file1,head1,head2,head3);
       return i;
   }

   if(strcmpi(command,"copy_column")==0) {
       if(parse(predicate,file1,predicate)==0) return 0;
       strcpy(file1,strcat(strcpy(path_job,c_config("JOB")),file1));
       if(parse(predicate,head1,predicate)==0) return 0;
       if(parse(predicate,file2,head2)==0) return 0;
       strcpy(file2,strcat(strcpy(path_job,c_config("JOB")),file2));
```

```
    i=call_copy_column(file1,head1,file2,head2);
    return i;
}

if(strcmpi(command,"match_column")==0) {
    if(parse(predicate,key_col,predicate)==0) return 0;
    if(parse(predicate,file1,predicate)==0) return 0;
    strcpy(file1,strcat(strcpy(path_job,c_config("JOB")),file1));
    if(parse(predicate,head1,predicate)==0) return 0;
    if(parse(predicate,file2,head2)==0) return 0;
    strcpy(file2,strcat(strcpy(path_job,c_config("JOB")),file2));
    i=call_match_column(key_col,file1,head1,file2,head2);
    return i;
}

if(strcmpi(command,"insert_formulae")==0) {
    if(parse(predicate,key_col2,predicate)==0) return 0;
    if(parse(predicate,file1,predicate)==0) return 0;
    strcpy(file1,strcat(strcpy(path_job,c_config("JOB")),file1));
    if(parse(predicate,key_col,predicate)==0) return 0;
    if(parse(predicate,head1,predicate)==0) return 0;
    if(parse(predicate,head2,operator)==0) return 0;
    i=call_insert_formulae(key_col2,file1,key_col,head1,head2,operator);
    return i;
}

if(strcmpi(command,"column_sum")==0) {
    if(parse(predicate,file1,predicate)==0) return 0;
    strcpy(file1,strcat(strcpy(path_job,c_config("JOB")),file1));
    if(parse(predicate,head1,head2)==0) return 0;
    i=call_column_sum(file1,head1,head2);
    return i;
}

if(strcmpi(command,"match_list")==0) {
    if(parse(predicate,file1,predicate)==0) return 0;
    strcpy(file1,strcat(strcpy(path_job,c_config("JOB")),file1));
    if(parse(predicate,head1,predicate)==0) return 0;
    if(parse(predicate,head2,predicate)==0) return 0;
    if(parse(predicate,head3,predicate)==0) return 0;
    if(parse(predicate,file2,predicate)==0) return 0;
    strcpy(file2,strcat(strcpy(path_job,c_config("JOB")),file2));
    if(parse(predicate,key_col,file3)==0) return 0;
    strcpy(file3,strcat(strcpy(path_job,c_config("JOB")),file3));
    i=call_match_list(file1,head1,head2,head3,file2,key_col,file3);
    return i;
}

if(strcmpi(command,"match_matrix")==0) {
    if(parse(predicate,key_col,predicate)==0) return 0;
    if(parse(predicate,file1,predicate)==0) return 0;
    strcpy(file1,strcat(strcpy(path_job,c_config("JOB")),file1));
    if(parse(predicate,head1,predicate)==0) return 0;
    if(parse(predicate,head2,predicate)==0) return 0;
    if(parse(predicate,file2,predicate)==0) return 0;
    strcpy(file2,strcat(strcpy(path_job,c_config("JOB")),file2));
    i=call_match_matrix(key_col,file1,head1,head2,file2);
    return i;
}
```

```
    if(strcmpi(command,"mult_vtM")==0) {
        if(parse(predicate,file1,predicate)==0) return 0;
        strcpy(file1,strcat(strcpy(path_job,c_config("JOB")),file1));
        if(parse(predicate,head1,predicate)==0) return 0;
        if(parse(predicate,head2,predicate)==0) return 0;
        if(parse(predicate,head3,predicate)==0) return 0;
        if(parse(predicate,head4,predicate)==0) return 0;
        i=call_mult_vtM(file1,head1,head2,head3,head4);
        return i;
    }

    if(strcmpi(command,"row_sum")==0) {
        if(parse(predicate,file1,predicate)==0) return 0;
        strcpy(file1,strcat(strcpy(path_job,c_config("JOB")),file1));
        if(parse(predicate,head1,predicate)==0) return 0;
        if(parse(predicate,head2,predicate)==0) return 0;
        if(parse(predicate,head3,predicate)==0) return 0;
        i=call_row_sum(file1,head1,head2,head3);
        return i;
    }

    printf("Unknown Command \n");
    return 0;
}

/* Control module for branching IF commands */

callif(char* cline)
{
    int i;
    char path_job[80];
    char command[80], predicate[80];
    char file[80], head[80];



    if(parse(cline,command,predicate)==0) return 0;

    if(strcmpi(command,"exists_file")==0) {
        strcpy(predicate,strcat(strcpy(path_job,c_config("JOB")),predicate));
        i=if_exists_file(predicate);
        return i;
    }

    if(strcmpi(command,"exists_column")==0) {
        if(parse(predicate,file,head)==0) return 0;
        strcpy(file,strcat(strcpy(path_job,c_config("JOB")),file));
        i=if_exists_column(file,head);
        return i;
    }

    if(strcmpi(command,"user_branch")==0) {
        i=if_user_branch(predicate);
        return i;
    }

    printf("Command Unknown \n");
    return 0;
}
```

RULE.DAT

```
call message CONFIRM EMPTY OUTPUT FILE
call edit_file to.wk1
call message CONFIRM OPERATIONS DATABASE FILE
call edit_file opertn.wk1
call message CONFIRM COST DATABASE FILE
call edit_file cost.wk1
call message CONFIRM PROJECT DATABASE FILE
call edit_file joblist.wk1
call comment BEGIN COST ESTIMATION SEQUENCE
call comment OPEN OUTPUT FILE
call copy_file to.wk1 work.wk1
call comment COPY OPERATIONS LIST TO OUTPUT FILE
call copy_column opertn.wk1 opertn work.wk1 opertn
call edit_file work.wk1
call comment MATCH PROJECT LIST AND OPERATIONS DATA TO OUTPUT FILE
call input_heading work.wk1 MARK1 * *
call match_list joblist.wk1 mem_id type number work.wk1 opertn opertn.wk1
call input_heading work.wk1 MARK2 * *
call edit_file work.wk1
call comment TABULATE TOTAL OPERATIONS
call row_sum work.wk1 MARK1 MARK2 ops
rem insert_formulae opertn work.wk1 ops MARK1 MARK2 @SUM(MARK1..MARK2)
call edit_file work.wk1
call comment MATCH UNIT COSTS WITH OPERATION TOTALS
call match_column opertn cost.wk1 cost work.wk1 ucst
call edit_file work.wk1
call comment CALCULATE TOTAL COST OF EACH OPERATION
call insert_formulae opertn work.wk1 tcst ops ucst ops*ucst
call edit_file work.wk1
call comment SUM COLUMN FOR TOTAL COST OF OPERATIONS
call column_sum work.wk1 mean tcst
call edit_file work.wk1
call comment MATCH COVARIANCE MATRIX TO OPERATION IN OUTPUT FILE
call input_heading work.wk1 MARK3 * *
call match_matrix opertn cost.wk1 MARK1 MARK2 work.wk1
call input_heading work.wk1 MARK4 * *
call edit_file work.wk1
call comment CALCULATE UNIT VARIANCE
call mult_vtM work.wk1 ops MARK3 MARK4 uvar
call edit_file work.wk1
call comment CALCULATE TOTAL VARIANCE OF EACH OPERATION
call insert_formulae opertn work.wk1 tvar ops uvar ops*uvar
call edit_file work.wk1
call comment SUM COLUMN FOR TOTAL VARIANCE
call column_sum work.wk1 variance tvar
call edit_file work.wk1
call comment REVIEW FINAL OUTPUT FILE
call edit_file work.wk1
call comment SUCCESS
```

N_CALL.H

```
/*********************************************************/
/*                                                       */
/*   This is the header file to n_call.c                 */
/*                                                       */
/*********************************************************/

#include "n_wksf.h"
#include <conio.h>
#include <process.h>
#include <dos.h>

#include "c_config.h"

int call_comment(char* predicate);
int call_message(char* predicate);
int call_copy_file(char* from, char* to);
int call_edit_file(char* predicate);
int call_input_heading(char *, char *, char *, char *);
int call_copy_column(char *, char *, char *, char *);
int call_match_column(char *, char *, char *, char *, char *);
int do_match_column(char*, char *, char *, char *, int, int);
int call_insert_formulae(char *,char *, char *, char *, char *, char *);
int call_column_sum(char *, char *, char *);
int call_exists_file(char *);
int call_match_list(char *, char *, char *, char *, char *, char *, char *);
int call_match_matrix(char *, char *, char *, char *, char *);
int call_mult_vtM(char *, char *, char *, char *, char *);
int call_row_sum(char *, char *, char *, char *);
int find_headings(lfile *,char *, char);
int find_empty_column(lfile *,char *);
int find_named_column(lfile *,char *,char *);
lfile *order_cell(lfile *,lfile *,char*, int, int);
int match_key(lfile *,char *, char *, int, int);
/* int sort_file(char *); */
lfile *find_cell(lfile *,char*, int, int);
lfile *fast_cell(lfile *, int, int);
char* decod_key(lfile *);
double decod_num(lfile *);
/* int wstar(char *, int); */
double decod_number(lfile *);
lfile *order_integer(lfile *,char *, int, int, int);
lfile *order_number(lfile *,char *, int, int, double);
lfile *order_string(lfile *,char *,int, int, char *);
lfile *order_formulae(lfile *,char *, int, int, char *);
char* replace_token(char *, char *, char *);
int write_formulae(char *, int, int, char *);
```

N_CALL.C

```
/*********************************************************/
/*                                                       */
/*   This file contains modules accessed by the          */
/*   inference engine through direct CALL commands.      */
/*                                                       */
/*********************************************************/

#include "n_call.h"

/* This routine prints out messages (w/o pausing) to the operator */

int call_comment(char* predicate)
{
   if(printf("\n %s \n",predicate)==0) return 0;
   return 1;
}

/* This routine prints out messages(w/ pausing) to the operator */

int call_message(char* predicate)
{
   if(printf("\n %s \n",predicate)==0) return 0;
   printf("Press any key to continue");
   getch();
   printf("\n");
   return 1;
}

/* This routine creates a copy wks_file from  named wks_file to */

int call_copy_file(char* from, char* to)
{
   lfile *wks_fp;

   wks_fp=(lfile *) malloc(sizeof(lfile));
   if(wks_fp==(lfile *) NULL) return 0;

   wks_fp=wksopen(wks_fp,from,to);
   if(wks_fp == (lfile *) NULL) {
       return 0;
   }

   wksclose(wks_fp);
   free(wks_fp);
   return 1;
}

/* This routine opens.wks_file file and allows it to be editted or reviewed. */

int call_edit_file(char* file)
{
   int status;
   char path_cntrl[80];
   char *args[2];
   char* message="Fatal Error\n";
   char name1[50];

   sprintf(name1,"VIP_CALC.EXE");
   name1[12]='\0';
```

```
    args[0]=name1;
    args[1]=file;
    args[2]=NULL;

    strcpy(path_cntrl,c_config("CNTRL"));
    strcat(path_cntrl,name1);
    strcat(path_cntrl,args[0]);

    status=spawnv(P_WAIT,path_cntrl,args);
    if(status==-1)perror(message);

    return 1;
}

/* This routine copies a column col1 in file1 to file2 and renames it col2 */

int call_copy_column(char* file1,char* col1,char* file2, char* col2)
{
    int row_from, col_from, row_to, col_to, i;
    int rowff, rowtt, colff, coltt;
    lfile *wks_fp, *wks_find, *wks_test;
    char head1[80], head2[80], head3[80];

    for(i=0;i<80;i++) head1[i]='\0';
    for(i=0;i<80;i++) head2[i]='\0';
    for(i=0;i<80;i++) head3[i]='\0';

    wks_fp=(lfile *) malloc(sizeof(lfile));
    if(wks_fp==(lfile *) NULL) return 0;

    col_from=find_named_column(wks_fp,file1,col1);
    if(col_from==-1) return 0;

    col_to=find_empty_column(wks_fp,file2);
    if(col_to==-1) return  0;

    row_from=find_headings(wks_fp,file1,'!');
    if(row_from==-1) return 0;

    wks_find=(lfile *) malloc(sizeof(lfile));
    if(wks_find==(lfile *) NULL) return 0;

    wks_find=wksopen(wks_find,file1,"");
    if(wks_find==(lfile *) NULL) return 0;

    strcpy(head1,col2);

    row_from=find_headings(wks_fp,file1,'@');
    if(row_from==-1) return 0;

    wks_test=(lfile *) fast_cell(wks_find,row_from,col_from);
    if(wks_test==(lfile *) NULL) strcpy(head2,"*");
    if(wks_test!=(lfile *) NULL) {
        strcpy(head2,decod_key(wks_test));
        wks_find=wks_test;
    }

    row_from=find_headings(wks_fp,file1,'#');
    if(row_from==-1) return 0;

    row_to=find_headings(wks_fp,file2,'#');
    if(row_to==-1) return 0;
```

```
    wks_test=(lfile *) fast_cell(wks_find,row_from,col_from);
    if(wks_test==(lfile *) NULL) strcpy(head3,"*");
    if(wks_test!=(lfile *) NULL) {
        strcpy(head3,decod_key(wks_test));
        wks_find=wks_test;
    }
    wksclose(wks_find);

    if(call_input_heading(file2,head1,head2,head3)==0) return 0;

    wks_find=wksopen(wks_find,file1,"");
    if(wks_find==(lfile *) NULL) return 0;

    for(i=2;;i++) {
        rowff=row_from+i;
        rowtt=row_to+i;
        colff=col_from;
        coltt=col_to;

        wks_test = (lfile *) fast_cell(wks_find,rowff,colff);
        if(wks_test==(lfile *) NULL) break;
        wks_find=wks_test;

        wks_fp=order_cell(wks_fp,wks_find,file2,rowtt,coltt);
        if(wks_fp==(lfile *) NULL) return 0;
    }
    wksclose(wks_find);
    free(wks_fp);
    free(wks_find);
    return 1;
}

/* This routine returns a pointer to a cell(row_find,col_find) in file1 */
/* It searches the entire file and is not dependent on order. */

lfile *find_cell(lfile *wks_fp,char* file1,int row_find, int col_find)
{
    int err;

    wks_fp=wksopen(wks_fp,file1,"");
    if(wks_fp==(lfile *) NULL) return (lfile *) NULL;

    for(;;) {
        err=wksnextc(wks_fp);
        if(err==-1) {
            wksclose(wks_fp);
            return (lfile *) NULL;
        }


        if(wks_fp->record.type.cell.row==row_find) {
            if(wks_fp->record.type.cell.column==col_find) {
                wksclose(wks_fp);
                return (lfile *) wks_fp;
            }
        }

        if(wks_fp->record.type.cell.row>row_find) {
            if(wks_fp->record.type.cell.column>col_find) {
                wksclose(wks_fp);
                return (lfile *) NULL;
            }
```

```
        }
    }
}

/* This routine has the same purpose as find_cell */
/* It searches an all ready open file in the forward direction only */
/* It is faster than find_cell, but can not backtrack */

lfile *fast_cell(lfile *wks_fp,int row_find, int col_find)
{
    int err;

    for(;;) {
        err=wksnextc(wks_fp);
        if(err==-1) {
            return (lfile *) NULL;
        }

        if(wks_fp->record.type.cell.row==row_find) {
            if(wks_fp->record.type.cell.column==col_find) {
                return (lfile *) wks_fp;
            }
        }

        if(wks_fp->record.type.cell.row>row_find) {
            if(wks_fp->record.type.cell.column>col_find) {
                return (lfile *) NULL;
            }
        }
    }
}

/* This routine uses a list in file1 composed of three headers super1/2/3 to generate */
/* a series of columns in file3 matched witha key_column in file3 */

int call_match_list(char* file1, char *super1, char *super2, char *super3, char* file2, char*
key_col, char* file3)
{
    int i;
    int row_from_head3;
    int col_from_head1, col_from_head2, col_from_head3;
    int row_to_head1, row_to_head2, row_to_head3;
    int col_to;
    int rowff, colff;
    lfile *wks_fp;
    lfile *wks_fp1, *wks_fp2, *wks_test1, *wks_test2, *wks_fp3, *wks_test3;
    char head1[80], head2[80], head3[80];

    for(i=0;i<80;i++) head1[i]='\0';
    for(i=0;i<80;i++) head2[i]='\0';
    for(i=0;i<80;i++) head3[i]='\0';

    wks_fp= (lfile *) malloc(sizeof(lfile));
    if(wks_fp==(lfile *) NULL) return 0;

    col_from_head1=find_named_column(wks_fp,file1,super1);
    if(col_from_head1==-1) return 0;

    col_from_head2=find_named_column(wks_fp,file1,super2);
    if(col_from_head2==-1) return 0;
```

```
col_from_head3=find_named_column(wks_fp,file1,super3);
if(col_from_head3==-1) return 0;

row_from_head3=find_headings(wks_fp,file1,'#');
if(row_from_head3==-1) return 0;

col_to=find_empty_column(wks_fp,file2);
if(col_to==-1) return  0;

row_to_head1=find_headings(wks_fp,file2,'!');
if(row_to_head1==-1) return 0;

row_to_head2=find_headings(wks_fp,file2,'@');
if(row_to_head2==-1) return 0;

row_to_head3=find_headings(wks_fp,file2,'#');
if(row_to_head3==-1) return 0;

wks_fp1=(lfile *) malloc(sizeof(lfile));
if(wks_fp1==(lfile *) NULL) return 0;

wks_fp1=wksopen(wks_fp1,file1,"");
if(wks_fp1==(lfile *) NULL) return 0;

wks_fp2=(lfile *) malloc(sizeof(lfile));
if(wks_fp2==(lfile *) NULL) return 0;

wks_fp2=wksopen(wks_fp2,file1,"");
if(wks_fp2==(lfile *) NULL) return 0;

wks_fp3=(lfile *) malloc(sizeof(lfile));
if(wks_fp3==(lfile *) NULL) return 0;

wks_fp3=wksopen(wks_fp3,file1,"");
if(wks_fp3==(lfile *) NULL) return 0;


for(i=2;;i++) {
    rowff=row_from_head3+i;
    colff=col_from_head1;

    wks_test1 = (lfile *) fast_cell(wks_fp1,rowff,colff);
    if(wks_test1==(lfile *) NULL) break;
    wks_fp1=wks_test1;

    strcpy(head1,decod_key(wks_fp1));
    if(head1[0]=='\0') head1[0]='*';

    rowff=row_from_head3+i;
    colff=col_from_head2;

    wks_test2 = (lfile *) fast_cell(wks_fp2,rowff,colff);
    if(wks_test2==(lfile *) NULL) return 0;
    wks_fp2=wks_test2;

    strcpy(head2,decod_key(wks_fp2));
    if(head2[0]=='\0') head2[0]='*';

    rowff=row_from_head3+i;
    colff=col_from_head3;

    wks_test3 = (lfile *) fast_cell(wks_fp3,rowff,colff);
    if(wks_test3==(lfile *) NULL) return 0;
    wks_fp3=wks_test3;
```

```
        strcpy(head3,decod_key(wks_fp3));
        if(head3[0]=='\0') head3[0]='*';

        col_to=find_empty_column(wks_fp,file2);
        if(call_input_heading(file2,head1,head2,head3)==0) return 0;
        if(do_match_column(key_col,file3,head2,file2, col_to, atoi(head3))==0) return 0;
    }

    wksclose(wks_fp1);
    wksclose(wks_fp2);
    wksclose(wks_fp3);
    free(wks_fp);
    free(wks_fp1);
    free(wks_fp2);
    free(wks_fp3);
    return 1;
}

/* This routine sorts the data found in col1 of file1 into col2 of file2 using key_col in */
/* both files as a reference */

int call_match_column(char* key_col,char* file1, char* col1, char* file2, char* col2)
{
    int i;
    int row_from_head1, row_from_head2, row_from_head3;
    int col_from, col_to;
    lfile *wks_fp, *wks_test;
    char head1[80], head2[80], head3[80];

    for(i=0;i<80;i++) head1[i]='\0';
    for(i=0;i<80;i++) head2[i]='\0';
    for(i=0;i<80;i++) head3[i]='\0';

    wks_fp=(lfile *) malloc(sizeof(lfile));
    if(wks_fp==(lfile *) NULL) return 0;

    row_from_head1=find_headings(wks_fp,file1,'!');
    if(row_from_head1==-1) return 0;

    row_from_head2=find_headings(wks_fp,file1,'@');
    if(row_from_head2==-1) return 0;

    row_from_head3=find_headings(wks_fp,file1,'#');
    if(row_from_head3==-1) return 0;

    col_from=find_named_column(wks_fp,file1,col1);
    if(col_from==-1) return 0;

    wks_fp=wksopen(wks_fp,file1,"");
    if(wks_fp==(lfile *) NULL) return 0;

    wks_test = (lfile *) fast_cell(wks_fp,row_from_head1,col_from);
    if(wks_test==(lfile *) NULL) return 0;
    wks_fp=wks_test;

    strcpy(head1,col2);
    if(head1[0]=='\0') head1[0]='*';

    wks_test = (lfile *) fast_cell(wks_fp,row_from_head2,col_from);
    if(wks_test!=(lfile *) NULL) {
        wks_fp=wks_test;
```

```
        strcpy(head2,decod_key(wks_fp));
    }
    if(wks_test==(lfile *) NULL) head2[0]='*';

    wks_test = (lfile *) fast_cell(wks_fp,row_from_head3,col_from);
    if(wks_test!=(lfile *) NULL) {
        wks_fp=wks_test;
        strcpy(head3,decod_key(wks_fp));
    }
    if(wks_test==(lfile *) NULL)  head3[0]='*';

    wksclose(wks_fp);

    col_to=find_empty_column(wks_fp,file2);
    if(call_input_heading(file2,head1,head2,head3)==0) return 0;
    if(do_match_column(key_col,file1,col1,file2,col_to,atoi(head3))==0) return 0;

    free(wks_fp);
    return 1;
}

/* This routine matches the actual column data in all match commands */

int do_match_column(char *key_col,char *file1,char *col1,char *file2, int col_to, int multiplier)
{
    int i;
    int key_from, key_to;
    int col_from, row_to;
    int colff, coltt, rowff, rowtt;
    int row_from_head3;
    lfile *wks_test, *wks_match, *wks_fp, *wks_order;
    char key[80];

    strnset(key,'\0',50);

    wks_fp=(lfile *) malloc(sizeof(lfile));
    if(wks_fp==(lfile *) NULL) return 0;

    wks_match=(lfile *) malloc(sizeof(lfile));
    if(wks_match==(lfile *) NULL) return 0;

    wks_order=(lfile *) malloc(sizeof(lfile));
    if(wks_order==(lfile *) NULL) return 0;

    key_from=find_named_column(wks_fp,file1,key_col);
    if(key_from==-1) return 0;

    row_from_head3=find_headings(wks_fp,file1,'#');
    if(row_from_head3==-1) return 0;

    col_from=find_named_column(wks_fp,file1,col1);
    if(col_from==-1) return 0;

    row_to=find_headings(wks_fp,file2,'#');
    if(row_to==-1) return 0;
    row_to=row_to+1;

    key_to=find_named_column(wks_fp,file2,key_col);
    if(key_to==-1) return 0;

    wks_fp=wksopen(wks_fp,file1,"");
    if(wks_fp==(lfile *) NULL) return 0;

    if(multiplier==0) multiplier=1;
```

```
    for(i=2;;i++) {

        colff=key_from;
        rowff=row_from_head3+i;

        wks_test=(lfile *) fast_cell(wks_fp,rowff,colff);
        if(wks_test==(lfile *) NULL) break;
        wks_fp=wks_test;

        strcpy(key,decod_key(wks_fp));

        rowff=rowff;
        colff=col_from;
        wks_test = (lfile *) fast_cell(wks_fp,rowff,colff);
        if(wks_test==(lfile *) NULL) return 0;
        wks_fp=wks_test;

        coltt=key_to;
        rowtt=row_to;
        rowtt=match_key(wks_match,file2,key,rowtt,coltt);
        if(rowtt!=-1) {

             coltt=col_to;
             rowtt=rowtt;

             if(multiplier==1) {
                  wks_match=order_cell(wks_match,wks_fp,file2,rowtt,coltt);
                  if(wks_match==(lfile *) NULL) return 0;
             }
             if(multiplier!=1) {
                  strcpy(key,decod_key(wks_fp));
                  wks_match=order_number(wks_order,file2,rowtt,coltt,multiplier*atof(key));
                  if(wks_match==(lfile *) NULL) return 0;
             }
        }
    }
    wksclose(wks_fp);
    free(wks_match);
    free(wks_fp);
    free(wks_order);

    return 1;
}

/* This routine decodes a worksheet cell pointer and returns a string */

char* decod_key(lfile *wks_fp)
{
    int err, integer;
    double number;
    char type[5], string[80];

    err=0;


    strnset(type,'\0',5);
    strnset(string,'\0',80);

    integer=0;
    number=0;

    err=wksdecod(wks_fp,type,&integer,&number,string);
    if(err==-1) return NULL;
```

```
   if(type[0]=='i') strcpy(string,itoa(integer,string,10));
   if(type[0]=='d') strcpy(string,gcvt(number,20,string));
   return string;
}

/* This routine returns the row number of a cell in a particular column of */
/* file which is identical to string key */

int match_key(lfile *wks_fp,char* file,char* key,int row_start,int col_start)
{
   char element[50];
   int col, row, i;
   lfile *wks_test;

   strnset(element,'\0',50);

   wks_fp=wksopen(wks_fp,file,"");
   if(wks_fp==(lfile *) NULL) return -1;

   for(i=1;;i++) {
       col=col_start;
       row=row_start+i;

       wks_test=fast_cell(wks_fp,row,col);
       if(wks_test==(lfile *) NULL) return -1;
       wks_fp=wks_test;

       strcpy(element,decod_key(wks_fp));

       if(strcmpi(element,key)==0) break;

   }

   wksclose(wks_fp);
   return row;
}

/* This routine inserts three column headings into the next available column of file */

int call_input_heading(char* file, char* head1, char* head2, char* head3)
{
   int row, col, i, j, k;
   lfile *wks_fp;

   wks_fp=(lfile *) malloc(sizeof(lfile));
   if(wks_fp==(lfile *) NULL) return 0;

   col = find_empty_column(wks_fp,file);
   if(col==-1) return 0;

   i=j=k=1;

   if(head1[0]!='*') {
       row = find_headings (wks_fp,file,'!');
       if(row==-1) return 0;
       wks_fp=order_string(wks_fp,file,row,col,head1);
       if(wks_fp==(lfile *) NULL) i=0;
       if(wks_fp!=(lfile *) NULL) i=1;
   }
```

```
    if(head2[0]!='*') {
        row = find_headings (wks_fp,file,'@');
        if(row==-1) return 0;
        wks_fp=order_string(wks_fp,file,row,col,head2);
        if(wks_fp==(lfile *) NULL) j=0;
        if(wks_fp!=(lfile *) NULL) j=1;
    }

    if(head3[0]!='*') {
        row = find_headings (wks_fp,file,'#');
        if(row==-1) return 0;
        wks_fp=order_number(wks_fp,file,row,col,atoi(head3));
        if(wks_fp==(lfile *) NULL) k=0;
        if(wks_fp!=(lfile *) NULL) k=1;
    }

    free(wks_fp);

    if(i==1 && j==1 && k==1) return 1;
    return 0;

}

/* This returns the row number of the column marker headings (!,@,#) in  file1 */

int find_headings(lfile *wks_fp,char* file1, char marker)
{
    char key[50];
    int err, row;

    strnset(key,'\0',50);

    wks_fp=wksopen(wks_fp,file1,"");
    if(wks_fp==(lfile *) NULL) return -1;

    row=0;

    for(;;) {
        err=wksnextc(wks_fp);

        if(err==-1) {
             wksclose(wks_fp);
             row=-1;
             return row;
        }

        row=wks_fp->record.type.cell.row;

        if(wks_fp->record.type.cell.column==0) {

             strcpy(key,decod_key(wks_fp));

             if(key!=NULL) {
                  if(key[1]==marker) {
                    wksclose(wks_fp);
                    return row;
                  }
             }
        }
    }
}

/* This routine returns the first available column space on the heading row */
```

```
int find_empty_column(lfile *wks_fp,char* file1)
{
   int row_find, col, err;
   char key[50];

   strnset(key,'\0',50);

   row_find=find_headings(wks_fp,file1,'!');
   if(row_find==-1) return -1;

   wks_fp=wksopen(wks_fp,file1,"");
   if(wks_fp==(lfile *) NULL) return -1;

   col=0;

   for(;;) {
       err=wksnextc(wks_fp);
       if(err==-1) {
            wksclose(wks_fp);
            col=col+1;
            return col;
       }

       if(wks_fp->record.type.cell.row==row_find) {
            col=wks_fp->record.type.cell.column;

            strcpy(key,decod_key(wks_fp));
            if(key==NULL) {
                 wksclose(wks_fp);
                 return col;
            }
       }

       if(wks_fp->record.type.cell.row>row_find) {
            col=col+1;
            wksclose(wks_fp);
            return col;
       }
   }
}

/* This routine returns the column location of a specific heading */

int find_named_column(lfile *wks_fp,char* file, char* col_find)
{
   int row_find, col, err, test, i;
   char key_col[50];
   char key[50];

   strnset(key_col,'\0',50);
   strnset(key,'\0',50);

   row_find=find_headings(wks_fp,file,'!');
   if(row_find==-1) return -1;

   wks_fp=wksopen(wks_fp,file,"");
   if(wks_fp==(lfile *) NULL) return -1;

   col=0;

   strcpy(key_col,col_find);
   strrev(key_col);
```

```
    for(;;) {
        err=wksnextc(wks_fp);
        if(err==-1) {
             wksclose(wks_fp);
             return -1;
        }

        if(wks_fp->record.type.cell.row==row_find) {
             col=wks_fp->record.type.cell.column;

             strcpy(key,decod_key(wks_fp));
             strrev(key);

             test=1;

             for(i=0;i<(strlen(col_find));i++) {
                  if(tolower(key[i])!=tolower(key_col[i])) test=0;
             }

             if(test==1) {
                  wksclose(wks_fp);
                  return col;
             }
        }

        if(wks_fp->record.type.cell.row>row_find) {
             wksclose(wks_fp);
             return -1;
        }
    }
}

/* This routine inserts a formulae form consisting of two variables head1 and head2 */
/* The formulae is inserted into column head of file1 using key_col as a guide to */
/* determine how many rows of formulae are required */

int call_insert_formulae(char* key_col,char* file,char* head,char* column1, char* column2, char*
operator)
{
    int i;
    int row_head;
    int col_factor1, col_factor2;
    int col_key;
    char name_factor1[80], name_factor2[80];
    char replace1[80], replace2[80], form[80];
    int col_head;
    lfile *wks_fp, *wks_test, *wks_out;

    strnset(replace1,'\0',80);
    strnset(replace2,'\0',80);
    strnset(name_factor1,'\0',80);
    strnset(name_factor2,'\0',80);
    strnset(form,'\0',80);

    wks_fp=(lfile *) malloc(sizeof(lfile));
    if(wks_fp==(lfile *) NULL) return 0;

    row_head=find_headings(wks_fp,file,'!');
    if(row_head==-1) return 0;

    col_factor1=find_named_column(wks_fp,file,column1);
    if(col_factor1==-1) return 0;
```

```
    col_factor2=find_named_column(wks_fp,file,column2);
    if(col_factor2==-1) return 0;

    col_key=find_named_column(wks_fp,file,key_col);
    if(col_key==-1) return 0;

    col_head=find_empty_column(wks_fp,file);
    if(col_head==-1) return 0;

    wks_fp=order_string(wks_fp,file,row_head,col_head,head);
    if(wks_fp==(lfile *) NULL) return 0;

    free(wks_fp);

    name_factor1[0]='\0';
    name_factor2[0]='\0';

    wksacol(name_factor1,col_factor1);
    wksacol(name_factor2,col_factor2);

    wks_out=(lfile *) malloc(sizeof(lfile));
    if(wks_out==(lfile *) NULL) return 0;

    wks_fp=(lfile *) malloc(sizeof(lfile));
    if(wks_fp==(lfile *) NULL) return 0;

    row_head=find_headings(wks_fp,file,'#');
    if(row_head==-1) return 0;

    wks_fp=wksopen(wks_fp,file,"find.wk1");
    if(wks_fp==(lfile *) NULL) return 0;
    wksclose(wks_fp);

    wks_fp=wksopen(wks_fp,"find.wk1","");
    if(wks_fp==(lfile *) NULL) return 0;

    for(i=2;;i++) {

        wks_test=(lfile *) fast_cell(wks_fp,row_head+i,col_key);
        if(wks_test==(lfile *) NULL) break;
        wks_fp=wks_test;

        strcpy(replace1,name_factor1);
        strcpy(replace2,name_factor2);

        wksarow(replace1,row_head+i);
        wksarow(replace2,row_head+i);

        strcpy(form,operator);
        strcpy(form,replace_token(form,column1,replace1));
        strcpy(form,replace_token(form,column2,replace2));

        if(order_formulae(wks_out,file,(row_head+i),col_head,form)==(lfile *) NULL) return 0;
    }

    wksclose(wks_fp);
    free(wks_fp);
    free(wks_out);
    return 1;
}

/* This routinr replaces token names head1 and head2 in formulae with spreadsheet column names */
```

```
char *replace_token(char* form,char* column,char* replace)
{
   int i,j,k,test;
   int pos,size_col,size_op;
   char store1[80], store2[80];

   for(;;) {
       if(strstr(form,column)==NULL) break;

       size_col=strlen(column);
       size_op=strlen(form);
       for(i=0;i<size_op-size_col+1;i++) {
            pos=-1;
            test=1;
            for(j=0;j<size_col;j++) {
                 if(form[i+j]!=column[j]) {
                    test=0;
                    break;
                 }
            }
            if(test==1) {
                 pos=i;
                 break;
            }
       }

       for(k=0;k<80;k++) store1[k]='\0';
       for(k=0;k<80;k++) store2[k]='\0';

       if((pos)>0) strncpy(store1,form,pos);
       strcat(store1,replace);
       form=strrev(form);
       strncpy(store2,form,(size_op-(pos+size_col)));
       store2[size_op-(pos+size_col)]='\0';
       strrev(store2);
       strcat(store1,store2);
       strcpy(form,store1);
   }
   return form;
}

/* This routine totals column head of file and leaves an indicator name next to the total */

int call_column_sum(char* file, char *name, char *head)
{
   int k, col_head, row_head;
   lfile *wks_fp, *wks_test;
   char col_name[80], start_row[80], stop_row[80], form[80];
   char line[80], title[80], col_string[80];

   for(k=0;k<80;k++) col_name[k]='\0';
   for(k=0;k<80;k++) start_row[k]='\0';
   for(k=0;k<80;k++) stop_row[k]='\0';
   for(k=0;k<80;k++) form[k]='\0';
   for(k=0;k<80;k++) line[k]='\0';
   for(k=1;k<13;k++) line[k]='=';
   line[0]='"';
   for(k=0;k<80;k++) col_string[k]='\0';

   strcpy(title,name);
   strcpy(col_string,head);
```

```
    wks_fp=(lfile *) malloc(sizeof(lfile));
    if(wks_fp==(lfile *) NULL) return 0;

    row_head=find_headings(wks_fp,file,'#');
    if(row_head==-1) return 0;

    col_head=find_named_column(wks_fp,file,col_string);
    if(col_head==-1) return 0;

    free(wks_fp);

    col_name[0]='\0';
    wksacol(col_name,col_head);

    strcpy(start_row,col_name);
    strcpy(stop_row,col_name);

    wksarow(start_row,row_head+1);
    wksarow(stop_row,row_head+1);

    wks_fp=(lfile *) malloc(sizeof(lfile));
    if(wks_fp==(lfile *) NULL) return 0;

    wks_fp=wksopen(wks_fp,file,"");
    if(wks_fp==(lfile *) NULL) return 0;

    for(k=2;;k++) {

        wks_test=(lfile *) fast_cell(wks_fp,row_head+k,col_head);
        if(wks_test==(lfile *) NULL) break;
        wks_fp=wks_test;

        stop_row[0]='\0';
        strcpy(stop_row,col_name);
        wksarow(stop_row,row_head+k);

    }
    wksclose(wks_fp);


    sprintf(form,"@SUM(%s..%s)",start_row,stop_row);

    wks_fp=order_string(wks_fp,file,row_head+k,col_head,line);
    wks_fp=order_string(wks_fp,file,row_head+k+1,col_head-1,title);
    wks_fp=order_formulae(wks_fp,file,row_head+k+1,col_head,form);
    wks_fp=order_string(wks_fp,file,row_head+k+2,col_head,line);
    wks_fp=order_string(wks_fp,file,row_head+k+3,col_head,line);


    if(wks_fp==(lfile *) NULL) return 0;
    free(wks_fp);
    return 1;

}

/* This routine inserts cell_pointer wks_fp2 into (row,col) of file */

lfile *order_cell(lfile *wks_fp1,lfile *wks_fp2,char *file,int row, int col)
{
    int r,c;
    lfile *wks_old, *wks_new;
    int err;

    wks_old=(lfile *) malloc(sizeof(lfile));
    if(wks_old==(lfile *) NULL) return (lfile *) NULL;
```

```
wks_old=wksopen(wks_old,file,"");
if(wks_old==(lfile *) NULL) return (lfile *) NULL;

wks_new=(lfile *) malloc(sizeof(lfile));
if(wks_new==(lfile *) NULL) return (lfile *) NULL;

wks_new=wksopen(wks_new,"","new.wk1");
if(wks_new==(lfile *) NULL) return (lfile *) NULL;

for(;;) {
    err=wksnextc(wks_old);
    if(err==-1) {

         wks_new->record=wks_fp2->record;
         wks_new->record.type.cell.row=row;
         wks_new->record.type.cell.column=col;
         wkswrec(wks_new);

         wksclose(wks_old);
         wksclose(wks_new);
         free(wks_old);
         free(wks_new);

         wks_fp1=wksopen(wks_fp1,"new.wk1",file);
         if(wks_fp1==(lfile *) NULL) return (lfile *) NULL;
         wksclose(wks_fp1);
         return (lfile *) wks_fp1;
    }

    r=wks_old->record.type.cell.row;
    c=wks_old->record.type.cell.column;

    if(r>row) break;
    if(r>=row && c>=col) break;

    wks_new->record=wks_old->record;
    wkswrec(wks_new);

}


wks_new->record=wks_fp2->record;
wks_new->record.type.cell.row=row;
wks_new->record.type.cell.column=col;
wkswrec(wks_new);

for(;;) {


    r=wks_old->record.type.cell.row;
    c=wks_old->record.type.cell.column;

    wks_new->record=wks_old->record;
    wkswrec(wks_new);

    err=wksnextc(wks_old);
    if(err==-1) {
         wksclose(wks_old);
         wksclose(wks_new);
         free(wks_old);
         free(wks_new);
```

```
            wks_fp1=wksopen(wks_fp1,"new.wk1",file);
            if(wks_fp1==(lfile *) NULL) return (lfile *) NULL;
            wksclose(wks_fp1);
            return (lfile *) wks_fp1;
        }
    }
}

/* This routine writes an integer to (row,col) of file */

lfile *wks_integer(lfile *wks_fp, char* file, int row, int col, int integer)
{
    int r,c,err;
    lfile *wks_old, *wks_new;

    wks_old=(lfile *) malloc(sizeof(lfile));
    if(wks_old==(lfile *) NULL) return (lfile *) NULL;

    wks_old=wksopen(wks_old,file,"");
    if(wks_old==(lfile *) NULL) return (lfile *) NULL;

    wks_new=(lfile *) malloc(sizeof(lfile));
    if(wks_new==(lfile *) NULL) return (lfile *) NULL;

    wks_new=wksopen(wks_new,"","new.wk1");
    if(wks_new==(lfile *) NULL) return (lfile *) NULL;

    for(;;) {
        err=wksnextc(wks_old);
        if(err==-1) {
            wksint(wks_new,PROTECTED|DEFAULT,col,row,integer);
            wksclose(wks_old);
            wksclose(wks_new);
            free(wks_old);
            free(wks_new);

            wks_fp=wksopen(wks_fp,"new.wk1",file);
            if(wks_fp==(lfile *) NULL) return (lfile *) NULL;
            wksclose(wks_fp);
            return (lfile *) wks_fp;
        }

        r=wks_old->record.type.cell.row;
        c=wks_old->record.type.cell.column;

        if(r>row) break;
        if(r>=row && c>=col) break;

        wks_new->record=wks_old->record;
        wkswrec(wks_new);

    }

    wksint(wks_new,PROTECTED|DEFAULT,col,row,integer);

    for(;;) {


        r=wks_old->record.type.cell.row;
        c=wks_old->record.type.cell.column;

        wks_new->record=wks_old->record;
        wkswrec(wks_new);
```

```
        err=wksnextc(wks_old);
        if(err==-1) {
             wksclose(wks_old);
             wksclose(wks_new);
             free(wks_old);
             free(wks_new);

             wks_fp=wksopen(wks_fp,"new.wk1",file);
             if(wks_fp==(lfile *) NULL) return (lfile *) NULL;
             wksclose(wks_fp);
             return (lfile *) wks_fp;
        }
    }
}

/* This routine writes adouble to (row,col) of file */

lfile *order_number(lfile *wks_fp,char* file, int row, int col, double number)
{
    int r,c,err;
    lfile *wks_old, *wks_new;

    wks_old=(lfile *) malloc(sizeof(lfile));
    if(wks_old==(lfile *) NULL) return (lfile *) NULL;

    wks_old=wksopen(wks_old,file,"");
    if(wks_old==(lfile *) NULL) return (lfile *) NULL;

    wks_new=(lfile *) malloc(sizeof(lfile));
    if(wks_new==(lfile *) NULL) return (lfile *) NULL;

    wks_new=wksopen(wks_new,"","new.wk1");
    if(wks_new==(lfile *) NULL) return (lfile *) NULL;

    for(;;) {
        err=wksnextc(wks_old);
        if(err==-1) {
             wksnum(wks_new,PROTECTED|DEFAULT,col,row,number);
             wksclose(wks_old);
             wksclose(wks_new);
             free(wks_old);
             free(wks_new);

             wks_fp=wksopen(wks_fp,"new.wk1",file);
             if(wks_fp==(lfile *) NULL) return (lfile *) NULL;
             wksclose(wks_fp);
             return (lfile *) wks_fp;
        }

        r=wks_old->record.type.cell.row;
        c=wks_old->record.type.cell.column;

        if(r>row) break;
        if(r>=row && c>=col) break;

        wks_new->record=wks_old->record;
        wkswrec(wks_new);

    }

    wksnum(wks_new,PROTECTED|DEFAULT,col,row,number);

    for(;;) {
```

```
        r=wks_old->record.type.cell.row;
        c=wks_old->record.type.cell.column;

        wks_new->record=wks_old->record;
        wkswrec(wks_new);

        err=wksnextc(wks_old);
        if(err==-1) {
             wksclose(wks_old);
             wksclose(wks_new);
             free(wks_old);
             free(wks_new);

             wks_fp=wksopen(wks_fp,"new.wk1",file);
             if(wks_fp==(lfile *) NULL) return (lfile *) NULL;
             wksclose(wks_fp);
             return (lfile *) wks_fp;
        }
    }
}


/* This routine writes a string to (row,col) of file */

lfile *order_string(lfile *wks_fp,char* file, int row, int col,char* string)
{
    int r,c,err;
    lfile *wks_old, *wks_new;

    wks_old=(lfile *) malloc(sizeof(lfile));
    if(wks_old==(lfile *) NULL) return (lfile *) NULL;

    wks_old=wksopen(wks_old,file,"");
    if(wks_old==(lfile *) NULL) return (lfile *) NULL;

    wks_new=(lfile *) malloc(sizeof(lfile));
    if(wks_new==(lfile *) NULL) return (lfile *) NULL;

    wks_new=wksopen(wks_new,"","new.wk1");
    if(wks_new==(lfile *) NULL) return (lfile *) NULL;

    for(;;) {
        err=wksnextc(wks_old);
        if(err==-1) {
             wkslabel(wks_new,PROTECTED|DEFAULT,col,row,string);
             wksclose(wks_old);
             wksclose(wks_new);
             free(wks_old);
             free(wks_new);

             wks_fp=wksopen(wks_fp,"new.wk1",file);
             if(wks_fp==(lfile *) NULL) return (lfile *) NULL;
             wksclose(wks_fp);
             return (lfile *) wks_fp;
        }

        r=wks_old->record.type.cell.row;
        c=wks_old->record.type.cell.column;

        if(r>row) break;
        if(r>=row && c>=col) break;
```

```
        wks_new->record=wks_old->record;
        wkswrec(wks_new);
    }

    wkslabel(wks_new,PROTECTED|DEFAULT,col,row,string);

    for(;;) {


        r=wks_old->record.type.cell.row;
        c=wks_old->record.type.cell.column;

        wks_new->record=wks_old->record;
        wkswrec(wks_new);

        err=wksnextc(wks_old);
        if(err==-1) {
             wksclose(wks_old);
             wksclose(wks_new);
             free(wks_old);
             free(wks_new);

             wks_fp=wksopen(wks_fp,"new.wk1",file);
             if(wks_fp==(lfile *) NULL) return (lfile *) NULL;
             wksclose(wks_fp);
             return (lfile *) wks_fp;
        }
    }
}

/* This routine writes a formulae to (row,col) of file */

lfile *order_formulae(lfile *wks_fp,char* file,int row,int col,char *operation)
{
    int r,c,err;
    lfile *wks_old, *wks_new;

    wks_old=(lfile *) malloc(sizeof(lfile));
    if(wks_old==(lfile *) NULL) return (lfile *) NULL;

    wks_old=wksopen(wks_old,file,"");
    if(wks_old==(lfile *) NULL) return (lfile *) NULL;

    wks_new=(lfile *) malloc(sizeof(lfile));
    if(wks_new==(lfile *) NULL) return (lfile *) NULL;

    wks_new=wksopen(wks_new,"","new.wk1");
    if(wks_new==(lfile *) NULL) return (lfile *) NULL;

    for(;;) {
        err=wksnextc(wks_old);
        if(err==-1) {
             wksform(wks_new,PROTECTED|DEFAULT,col,row,operation);
             wksclose(wks_old);
             wksclose(wks_new);
             free(wks_old);
             free(wks_new);
             wks_fp=wksopen(wks_fp,"new.wk1",file);
             if(wks_fp==(lfile *) NULL) return (lfile *) NULL;
             wksclose(wks_fp);
             return (lfile *) wks_fp;
        }
```

```
        r=wks_old->record.type.cell.row;
        c=wks_old->record.type.cell.column;

        if(r>row) break;
        if(r>=row && c>=col) break;

        wks_new->record=wks_old->record;
        wkswrec(wks_new);

    }

    wksform(wks_new,PROTECTED|DEFAULT,col,row,operation);

    for(;;) {
        r=wks_old->record.type.cell.row;
        c=wks_old->record.type.cell.column;

        wks_new->record=wks_old->record;
        wkswrec(wks_new);

        err=wksnextc(wks_old);
        if(err==-1) {
             wksclose(wks_old);
             wksclose(wks_new);
             free(wks_old);
             free(wks_new);

             wks_fp=wksopen(wks_fp,"new.wk1",file);
             if(wks_fp==(lfile *) NULL) return (lfile *) NULL;
             wksclose(wks_fp);
             return (lfile *) wks_fp;
        }
    }
}

/* This routine matches a matrix between markers col1 and col2 in file1 to file2 using a key_col in
both */

int call_match_matrix(char* key_col,char* file1, char* col1, char* col2, char* file2)
{

    int i;
    int col_corner_from, size;
    int row_corner_to, col_corner_to, col_key_to;
    int row_from_head1, row_from_head2, row_from_head3;
    int col_from, col_to;
    lfile *wks_fp, *wks_test;
    char head1[80], head2[80], head3[80];

    for(i=0;i<80;i++) head1[i]='\0';
    for(i=0;i<80;i++) head2[i]='\0';
    for(i=0;i<80;i++) head3[i]='\0';

    wks_fp=(lfile *) malloc(sizeof(lfile));
    if(wks_fp==(lfile *) NULL) return 0;

    row_from_head1=find_headings(wks_fp,file1,'!');
    if(row_from_head1==-1) return 0;

    row_from_head2=find_headings(wks_fp,file1,'@');
    if(row_from_head2==-1) return 0;

    row_from_head3=find_headings(wks_fp,file1,'#');
    if(row_from_head3==-1) return 0;
```

```
    col_from=find_named_column(wks_fp,file1,col1);
    if(col_from==-1) return 0;

    col_corner_from=col_from+1;

    size=find_named_column(wks_fp,file1,col2);
    if(size==-1) return 0;
    size=size-col_corner_from;

    row_corner_to=find_headings(wks_fp,file2,'#');
    if(row_corner_to==-1) return 0;
    row_corner_to=row_corner_to+1;

    col_corner_to=find_empty_column(wks_fp,file2);
    if(col_corner_to==-1) return 0;

    col_key_to=find_named_column(wks_fp,file2,key_col);
    if(col_key_to==-1) return 0;

    for(i=0;i<size;i++) {
        wks_fp=wksopen(wks_fp,file1,"");
        if(wks_fp==(lfile *) NULL) return 0;

        wks_test = (lfile *) fast_cell(wks_fp,row_from_head1,col_corner_from+i);
        if(wks_test!=(lfile *) NULL) {
             wks_fp=wks_test;
             strcpy(head1,decod_key(wks_fp));
        }
        if(wks_test==(lfile *) NULL) return 0;

        wks_test = (lfile *) fast_cell(wks_fp,row_from_head2,col_corner_from+i);
        if(wks_test!=(lfile *) NULL) {
             wks_fp=wks_test;
             strcpy(head2,decod_key(wks_fp));
        }
        if(wks_test==(lfile *) NULL) head2[0]='*';

        wks_test = (lfile *) fast_cell(wks_fp,row_from_head3,col_corner_from+i);
        if(wks_test!=(lfile *) NULL) {
             wks_fp=wks_test;
             strcpy(head3,decod_key(wks_fp));
        }
        if(wks_test==(lfile *) NULL)  head3[0]='*';

        wksclose(wks_fp);

        col_to=match_key(wks_fp,file2,head1,row_corner_to,col_key_to);
        if(col_to==-1) return 0;
        col_to=col_to-(row_corner_to+1)+col_corner_to;


        if(call_input_heading(file2,head1,head2,head3)==0) return 0;
        if(do_match_column(key_col,file1,head1,file2,col_to,atoi(head3))==0) return 0;
    }

    free(wks_fp);
    return 1;
}

/* This routine performs matrix algebra vTM on a vector col1 and a matrix found */
/* between col2 and col3. The result is stored as col4 */
```

```
int call_mult_vtM(char* file1, char* col1,char* col2,char* col3,char* col4)
{
   int i, j, row_head3;
   int row_vector, col_vector;
   int row_matrix_corner, col_matrix_corner, matrix_size;
   int row_matrix, col_matrix;
   int row_new,col_new;
   double sum, vector_element, matrix_element;
   char string[80];
   lfile *wks_fp1, *wks_fp2;

   for(i=0;i<80;i++) string[i]='\0';

   wks_fp1=(lfile *) malloc(sizeof(lfile));
   if(wks_fp1==(lfile *) NULL) return 0;

   wks_fp2=(lfile *) malloc(sizeof(lfile));
   if(wks_fp2==(lfile *) NULL) return 0;

   row_head3=find_headings(wks_fp1,file1,'#');
   if(row_head3==-1) return 0;

   col_vector=find_named_column(wks_fp1,file1,col1);
   if(col_vector==-1) return 0;

   col_matrix_corner=find_named_column(wks_fp1,file1,col2);
   if(col_matrix_corner==-1) return 0;
   col_matrix_corner=col_matrix_corner+1;

   row_matrix_corner=row_head3+2;

   matrix_size=find_named_column(wks_fp1,file1,col3);
   if(matrix_size==-1) return 0;
   matrix_size=matrix_size-col_matrix_corner;

   call_input_heading(file1,col4,"*","*");

   col_new=find_named_column(wks_fp1,file1,col4);
   if(col_new==-1) return 0;

   for(i=0;i<matrix_size;i++){
       row_new=row_head3+i+2;
       col_matrix=col_matrix_corner+i;
       sum=0;

       wks_fp1=( lfile *) wksopen(wks_fp1,file1,"");
       if(wks_fp1==(lfile *) NULL) return 0;

       wks_fp2=( lfile *) wksopen(wks_fp2,file1,"");
       if(wks_fp2==(lfile *) NULL) return 0;

       for(j=0;j<matrix_size;j++) {
            row_vector=row_head3+2+j;
            row_matrix=row_matrix_corner+j;

            wks_fp1=(lfile *) fast_cell(wks_fp1,row_vector,col_vector);
            if(wks_fp1==(lfile *) NULL) return 0;
            strcpy(string,decod_key(wks_fp1));
            vector_element=atof(string);

            wks_fp2=(lfile *) fast_cell(wks_fp2,row_matrix,col_matrix);
            if(wks_fp2==(lfile *) NULL) return 0;
            strcpy(string,decod_key(wks_fp2));
            matrix_element=atof(string);
```

```
            sum=sum+vector_element*matrix_element;
        }

        wksclose(wks_fp1);
        wksclose(wks_fp2);

        wks_fp1=order_number(wks_fp1,file1,row_new,col_new,sum);
        if(wks_fp1==(lfile *) NULL) return 0;
    }

    free(wks_fp1);
    free(wks_fp2);
    return 1;

}

/* This routine sums row elements between col1 and col2, storing the result in col3 */

int call_row_sum(char* file1, char* col1,char* col2,char* col3)
{
    int i, j, row_head3;
    int row, col, size;
    int col_start, col_new;
    double row_element, sum;
    lfile *wks_fp, *wks_test;

    wks_fp=(lfile *) malloc(sizeof(lfile));
    if(wks_fp==(lfile *) NULL) return 0;

    row_head3=find_headings(wks_fp,file1,'#');
    if(row_head3==-1) return 0;

    col_start=find_named_column(wks_fp,file1,col1);
    if(col_start==-1) return 0;
    col_start=col_start+1;

    size=find_named_column(wks_fp,file1,col2);
    if(size==-1) return 0;
    size=size-col_start;

    call_input_heading(file1,col3,"*","*");

    col_new=find_named_column(wks_fp,file1,col3);
    if(col_new==-1) return 0;

    for(i=0;;i++){
        row=row_head3+2+i;
        sum=0;

        wks_fp=( lfile *) wksopen(wks_fp,file1,"");
        if(wks_fp==(lfile *) NULL) return 0;

        for(j=0;j<size;j++) {
            col=col_start+j;

            wks_test=(lfile *) fast_cell(wks_fp,row,col);
            if(wks_test==(lfile *) NULL) {
                wksclose(wks_fp);
                free(wks_fp);
                return 1;
            }
            wks_fp=wks_test;
            row_element=atof(decod_key(wks_fp));
```

```
            sum=sum+row_element;
        }

        wksclose(wks_fp);

        wks_fp=order_number(wks_fp,file1,row,col_new,sum);
        if(wks_fp==(lfile *) NULL) return 0;
    }

}
```

N_CAIF.H

```
/*********************************************************/
/*                                                       */
/* This is the header file to n_caif.c                   */
/*                                                       */
/*********************************************************/

#include "n_call.h"

int if_exists_file(char *);
int if_exists_column(char *,char *);
int if_user_branch(char *);
int call_exists_file(char *);
```

N_CAIF.C

```
/*********************************************************/
/*                                                       */
/* This file contains modules accessed by directly by    */
/* branching IF and IFNOT commands.                      */
/*                                                       */
/*********************************************************/

#include "n_caif.h"

#include <compiler.h>
#include <wks.h>

struct unit {
   char name[80];
   struct unit *next;
   struct unit *branch;
};

struct stack {
   struct unit *stack;
   struct stack *prev;
};
```

```
/* This routine establishes the existence of a particular spreadsheet file */

int if_exists_file(char* predicate)
{
   if(call_exists_file(predicate)==0) return -1;
   return 1;
}

/* This routine establishes the existence of a particular column head in file */

int if_exists_column(char* file,char* head)
{
   lfile *wks_fp;

   wks_fp=(lfile *) malloc(sizeof(lfile));
   if(wks_fp==(lfile *) NULL) return 0;

   if(find_named_column(wks_fp,file,head)==-1) {
       free(wks_fp);
       return -1;
   }

   free(wks_fp);
   return 1;
}

/* This routine allows user controlled branches */

int if_user_branch(char* predicate)
{
   int ch;

   ch='\0';

   for(;;) {
       printf("%s (Y/N)\n",predicate);
       ch=getch();

       if(ch=='Y' || ch=='y') {
            printf("\n");
            return 1;
       }
       if(ch=='N' || ch=='n') {
            printf("\n");
            return -1;
       }
   }
}

int call_exists_file(char *predicate)
{
   lfile *wks_fp;
   wks_fp=(lfile *) malloc(sizeof(lfile));
   if(wks_fp==(lfile *) NULL) {
       printf("OUT Of Mem/n");
       return 0;
   }
   if(wksopen(wks_fp,predicate,"")==(lfile *) NULL) return 0;
   wksclose(wks_fp);
   return 1;
}
```

N_FILE.H

```
/*********************************************************/
/*                                                       */
/*   This is the header file to n_file.c                 */
/*                                                       */
/*********************************************************/

#include <stdio.h>
#include <string.h>

#include "n_stck.h"

extern struct unit;
extern struct stack;


struct unit *rule_load(char* fname);
struct unit *new_unit(void);
struct stack *new_stack(void);
struct stack *push_on(struct stack *s, struct unit *u);
struct stack *pull_off(struct stack *s);
```

N_FILE.C

```
/*********************************************************/
/*                                                       */
/*   This file contains instructions for loading the     */
/*   rule base into a linked list structure              */
/*                                                       */
/*********************************************************/

#include "n_file.h"

struct unit {
   char name[80];
   struct unit *next;
   struct unit *branch;
};

struct stack {
   struct unit *stack;
   struct stack *prev;
};

struct unit *rule_load(char* fname)
{
   char name [80];
   int spc, oldspc, i;
   struct unit *u, *v, *start;
   struct stack *s;
   FILE *fp;
```

```
fp=fopen(fname,"r");
if(fp==0) return 0;

u=(struct unit *) new_unit();
if(!u) return 0;
start=u;

s=(struct stack *) new_stack();
if(!s) return 0;

oldspc=0;

for(;;) {
   spc=0;
   fgets(name,80,fp);
    if(name[0]==0) return start;
   for(i=0;i<sizeof(name);i++) {
        if(name[i]==' ') spc++;
        if(name[i]=='\n') return start;
        if(name[i]=='\0') return start;
        if(name[i]!=' ') break;
    }

    for(i=0;i<sizeof(name);i++) {
         if(name[i]=='\n') {
              name[i]='\0';
              break;
         }
    }

   if(spc>oldspc) {
    for(i=0;i<(spc-oldspc);i++) {
      s=(struct stack *) push_on(s,u);
      if(!s) {
         fclose(fp);
         return 0;
      }

      v=(struct unit *) new_unit();
      if(!v) {
         fclose(fp);
         return 0;
      }

      u->branch=v;
      u=v;
   }
   }

   if(spc==oldspc) {
   v=(struct unit *) new_unit();
   if(!v) return 0;
   u->next=v;
   u=v;
   }

   if(spc<oldspc) {
   for(i=0;i<(oldspc-spc);i++) {
      s=(struct stack *) pull_off(s);
      if(!s) return 0;
      u=s->stack;
   }
```

```
      v=(struct unit *) new_unit();
      if(!v) return 0;
      u->next=v;
      u=v;
      }

      oldspc=spc;
      for(i=spc;i<sizeof(name);i++) u->name[i-spc]=name[i];
   }
}
```

N_STCK.H

```
/*********************************************************/
/*                                                       */
/*   This is the header file for n_stck.c                */
/*                                                       */
/*********************************************************/

#include <stdio.h>
#include <string.h>
#include <alloc.h>

extern struct unit;
extern struct stack;


struct unit *new_unit(void);
struct stack *new_stack(void);
struct stack *push_on(struct stack *s, struct unit *u);
struct stack *pull_off(struct stack *s);
```

N_STCK.C

```
/*********************************************************/
/*                                                       */
/*   This file contains stack utilities for manipulating */
/*   the linked rule list                                */
/*                                                       */
/*********************************************************/

#include "n_stck.h"

struct unit {
   char name[80];
   struct unit *next;
   struct unit *branch;
};
```

```
struct stack {
   struct unit *stack;
   struct stack *prev;
};

struct unit un;
struct stack st;

/* Create a new rule unit */

struct unit *new_unit()
{
   struct unit *u;
   u=(struct unit *) malloc(sizeof(un));
   if(!u) return 0;

   u->next=NULL;
   u->branch=NULL;
   strset(u->name,' ');
   u->name[79]='\0';

   return u;
}

/*  Create a new stack unit */

struct stack *new_stack()
{
   struct stack *s;
   s=(struct stack *) malloc(sizeof(st));
   if(!s) return 0;

   s->stack=NULL;
   s->prev=NULL;

   return s;
}

/* Add another rule unit to the stack */

struct stack *push_on(struct stack *s, struct unit *u)
{

   struct stack *t;

   t=(struct stack *) new_stack();
   if(!t) return 0;

   s->stack=u;
   t->prev=s;
   s=t;

   return s;
}

/* Remove a rule unit from the stack */

struct stack *pull_off(struct stack *s)
{
   struct unit *u;
   struct stack *olds;

   olds=s;
```

```
    s=s->prev;
    if(!s) return 0;

    u=s->stack;
    if(!u) return 0;

    free(olds);

    return s;
}
```