A KNOWLEDGE-BASED SYSTEM FOR HIGHWAY LOCATION

by

BERNARDO DE CASTILHO

A THESIS SUBMITTED IN PARTIAL FULFILMENT 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

February 1987

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
2075 Wesbrook Place
Vancouver, Canada
V6T 1W5

Date: February 1987

ABSTRACT

Knowledge-based systems are increasingly popular in a variety of fields. These systems may provide an elegant and comprehensive framework for the solution of ill-structured, complex problems that traditional computer systems cannot handle.

This paper describes the implementation process of a prototype knowledge-based system designed to help in the task of locating and evaluating highway corridors, taking into account environmental, social and economic factors. The applicability of the approach to other engineering problems is also analyzed as well as the relative merits of alternate implementation tools.

The most significant aspects of the system presented are its transparent reasoning, provided by extensive explanation facilities, the original spatial data representation structures it uses and the implementation of fuzzy operators to deal with uncertain information. The system presented has a comparatively reduced knowledge base and therefore does not achieve the level of performance expected from experts in the field. However, it does demonstrate how relatively new Artificial Intelligence techniques can be used in conjunction with conventional techniques to model a typical Civil Engineering problem.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to thank Dr. Francis P. D. Navin for supervising this project and supplying information in the field of Transportation Engineering, and to Glen Cooper for the encouragement and support provided in the fascinating field of Artificial Intelligence. I have also benefited from useful discussion with Doug Perkins who generously shared some of his extensive expertise in highway location.

I would also like to express my gratitude to Ron Yaworsky for his invaluable help in almost every aspect of this project.

Finally, special thanks to Justin Williamson, Alejandro Allievi, Ibrahim Al Hamad, Karen Heiberg and all my friends at UBC who, directly or otherwise, made my work more challenging and rewarding.

1. INTRODUCTION

Recently developed techniques in the Artificial Intelligence field hold the promise of elegant and natural solutions to ill-structured, complex problems in many fields.

While there is abundant literature describing the general principles involved and existing systems, their domain-specific nature currently prevents off-the-shelf solutions from being useful except for relatively simple applications. As a result, most practical systems have to be especially designed to accommodate the unique characteristics of each particular problem.

The major objective of this study is to investigate the applicability of the knowledge-based system approach to highway corridor location. To achieve this, a prototypical rule-based system was implemented. The Highway Location Assistant system (HLA) incorporates many of the techniques that can be found in the literature and some that were developed especially for this particular application. The manner in which these procedures are combined into a complete system is also described, for this is a point of paramount importance on which literature is currently scarce.

HLA consists of several modules, some of which are purely knowledge-based and others that implement conventional procedures. The system has, therefore, a hybrid nature. This demonstrates that, rather than replacing existing tools and procedures,

knowledge-based modules can complement them and enhance their usefulness.

Two distinct topics are discussed in this paper: highway corridor location procedures and knowledge-based problem solving techniques. The major emphasis is placed on the latter. Highway corridor location is used as an example of a typical ill-structured civil engineering problem. The techniques currently employed to solve it are described to demonstrate the potential advantages of the new approach and to provide a necessary insight on how HLA actually works.

1.1. The Highway Corridor Location Problem

The process of selecting a highway corridor involves varying degrees of expertise in many fields, as well as political factors. Each possible route has associated economic, environmental and social impacts. These impacts can be positive or negative, and to describe each one it is necessary to assess its extent and importance.

There are standard procedures for the assessment of the extent of many types of impacts (e.g. construction and operating costs), but the relative importance of each of those values has to be determined by experts. Furthermore, some types of impacts (such as aesthetic quality) are subjective by nature and, therefore, difficult to assess and justify. As a result, only seasoned practitioners are able to use

existing routing systems. Most of these are useful engineering tools, but cannot provide a designer with advice or demonstrate how they obtained their results.

1.2. Knowledge-Based Systems

Computers have traditionally been used to solve problems that are formalized and analytical in structure. The requirement of explicit formalization of the problem into detailed, sequential statements has restricted the use of the computer to problems that have well understood, systematic solution procedures. The interest in expert system techniques was led by the desire to use the computer to aid in the solution of less structured, formalized problems.

The term "knowledge-based systems" refers to systems which contain a substantial amount of knowledge and help solve unstructured classes of problems without necessarily achieving expert level performance. It is preferred over the term "expert systems" in this paper because it encompasses a wider class of applications. Additionally, the latter term seems to give some people the wrong connotation that these systems will also learn as well as human experts do.

Domain independent problem solving strategies are commonly referred to as weak methods and may lead to combinatorial explosions while searching for solutions due to a potential lack of focus. Knowledge-based systems, on the other hand, can be

considered strong problem solvers, since they employ knowledge which is applicable to one specific field only (domain knowledge).

Current knowledge-based systems normally combine two basic problem solving approaches: derivation and formation. Derivation involves selecting a solution that is most appropriate for the problem at hand from a list of predefined solutions. This is the more common approach, supported by many commercial "expert system shells". It is accomplished by traversing a tree, using rules to select the appropriate branches and reaching the final solution at the bottom of the tree.

The formation approach involves forming a solution from eligible components stored in the system's knowledge base. It is more complex than the derivation approach, since it involves selecting solutions and combining them. The domain specific nature of this type of procedure makes most of the literature on the subject either very general or too specialized to be useful in other applications.

1.3. Advantages and Limitations

There are several reasons for the rapidly increasing popularity of knowledge-based systems:

Knowledge-based systems help solve ill-structured problems for which systematic

solution procedures do not exist.

They can make expertise and knowledge available on a twenty-four hour basis, whereas human experts are obviously less available. Furthermore, multiple replication can make this knowledge available over dispersed geographical areas. Rather than replacing experts, these systems usually allow them to allocate their time more efficiently.

One of the greatest benefits brought by such systems is the organization of knowledge. It has been observed in the past that the rigours of extracting and adapting knowledge to a computer system has enforced a discipline on the organization and documentation of knowledge.

By using these systems and the explanation facilities usually incorporated into them, technicians can gain new insights into the problem and, sometimes, emulate the performance of an expert.

Furthermore, knowledge that is captured in a computer system can be retained indefinitely. Human knowledge is a perishable asset, whereby experts can be lured away by other companies or institutions, or subject to accident or illness.

Of course these systems have many limitations that should be considered as well.

A common misconception that people have about knowledge-based systems is that

they can somehow provide "magical" approaches to solving complex problems. Knowledge-based systems can be thought of as models of the expertise of the practitioners available in the field. In order to create such models, it is obvious that there must be genuine human experts capable of performing the tasks.

Knowledge-based systems cannot be built quickly. Systems designed to solve moderately complex problems currently take five to ten person-years to develop. This time is being reduced, however, as more efficient techniques and development tools become available.

The central postulate of knowledge engineering is that systems achieve expert performance from rich, diverse knowledge bases rather than from clever algorithms. Therefore, the knowledge and reasoning methods used by the human practitioners have to be accessible. This is the greatest difficulty associated with the implementation of knowledge-based systems, since, generally, the more skilled the experts become in their fields, the less able they are to explain how they perform and use their skill. Experts tend to rely on unstructured, qualitative knowledge that can be described as "intuition". Still, significant insights may be gained from interviews with practitioners and from the literature of normative and descriptive theories.

Another difficulty associated with development of the knowledge base is the potential reluctance of skilled practitioners to divulge their "tricks of the trade", resulting from the perception that their preeminence in the field may be

jeopardized.

The domains in which these systems perform have to be well defined. If they are too broad, the magnitude of the task will be too great and development will become very difficult. On the other hand, if the specified domain is too limited, the usefulness of the system will be compromised.

1.4. Anatomy of a Typical Knowledge-Based System

The typical architecture of knowledge-based and conventional computer systems are noticeably different, as depicted in Figure 1. While most conventional systems are composed of a set of programs and the data they use as input, knowledge-based systems can often be logically divided into three parts: the inference engine, the knowledge base and the database. These parts are described in the following sections.

1.4.1. Inference Procedures

The inference engine module is relatively domain-independent. It contains enough logic to interpret the information in the knowledge base. In addition, it will usually provide explanation facilities, apply uncertainty propagation mechanisms and manage the database.

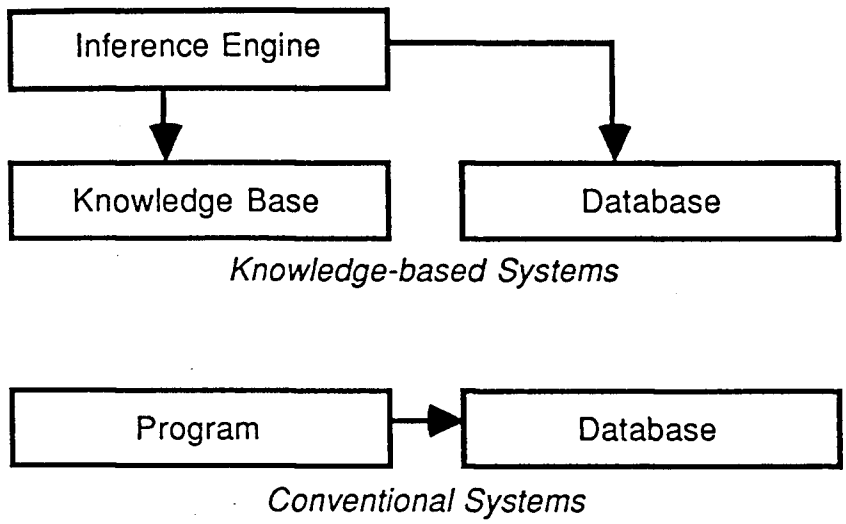


Figure 1. Different Architectures

Inference procedures are usually described as forward or backward chaining inference. Backward chaining, or goal-driven inference, is appropriate when there is a reasonable number of possible final answers to the problem under study. Classical examples are identification systems, such as programs that identify diseases based on symptoms or the chemical structure of substances based on their properties. It has been suggested that experts follow a goal-driven path to the solution of a problem. Figure 2 depicts a backward chaining, or goal-driven inference process: based on initial data, hypotheses are formulated and tested until an acceptable answer is found.

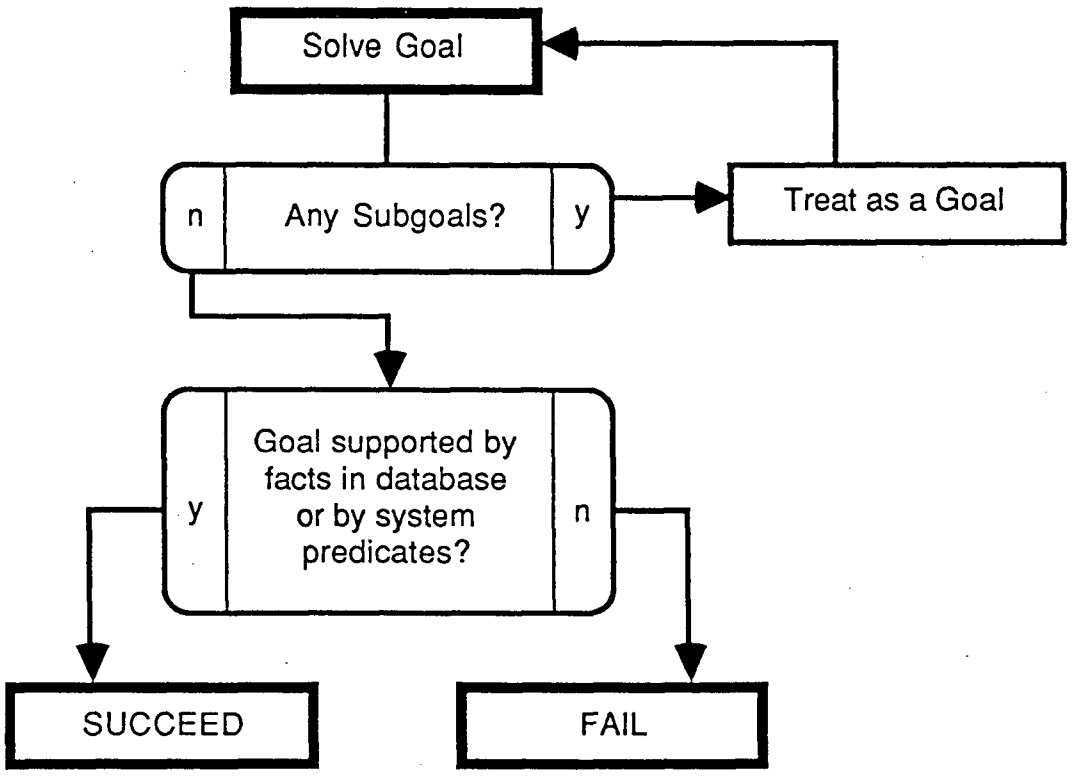


Figure 2. A Backward-Chaining Inference Engine

Forward chaining, or data-driven inference, is practical when combinatorial explosion creates a seemingly infinite number of potentially correct answers, such as possible configurations of machines. Systems that use this mechanism are sometimes called production systems. The process involves drawing conclusions from the available data, adding these conclusions to the database, checking if the expanded database provides a solution to the problem and possibly repeating these steps several times. Figure 3 depicts a forward-chaining process.

Many problems require a combination of these approaches (Mixed Initiative systems). Maher [25] demonstrated how a simple engineering problem can be solved using either method. Merrit [26] discussed how Prolog's native, backward chaining inference engine, can be used to effectively implement a data-driven inference system.

1.4.2. Explanation Capabilities

Knowledge-based systems are designed to solve complex and poorly structured problems that would otherwise have to be solved by a human practitioner rather than by conventional programs. Like human practitioners, these systems must pass through a relatively lengthy apprenticeship stage during which their knowledge bases are expanded and modified. Even after they have achieved expert levels of performance, they are capable, like all human experts, of making mistakes.

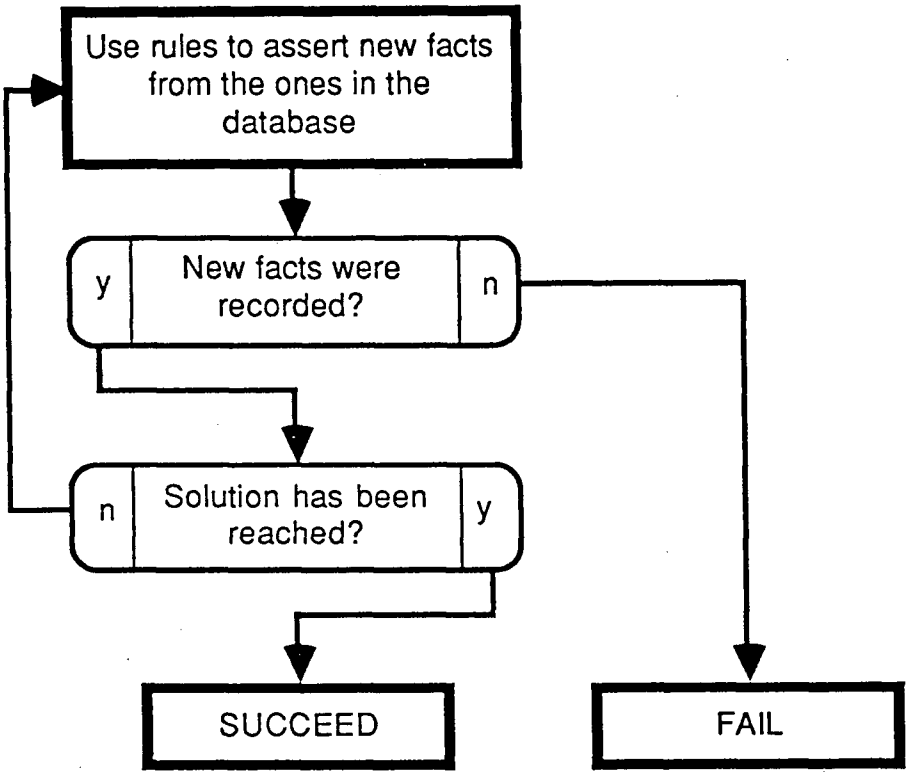


Figure 3. A Forward-Chaining Inference Engine

This is the basic reason why virtually all knowledge-based systems provide explanation facilities. By analyzing their reasoning, human experts and knowledge engineers can improve their performance.

At first glance it would seem that conventional systems have a distinct advantage over expert systems in this regard. However, this advantage is illusory. Conventional programs for performing complex tasks, like those suitable for knowledge-based systems, would be subject to the same mistakes. In addition, their mistakes would be very difficult to remedy since the strategies, heuristics and basic assumptions upon which these programs are based will not be explicitly stated in the program code [37].

1.4.3. Knowledge Representation

The knowledge base is usually the most time-consuming and critical module to develop. To be effective, it has to be able to represent a sufficient amount of problem-solving expertise pertinent to the domain of interest, in a consistent and relatively easy to expand manner.

The knowledge base consists of knowledge representation structures and, unlike the conventional database, contains general domain information, i.e., it is not restricted to any particular problem. Development of knowledge representation structures is currently a popular field of Artificial Intelligence research. The effectiveness of a

given structure depends on how the information it contains will be used and on the nature of the knowledge it represents. Some types of commonly used structures are briefly described below.

1.4.3.1. Predicate Calculus

Predicate Calculus requires that facts and relationships be presented as *IF-THEN* structures. Rules of inference operate upon these to generate new true facts. This type of knowledge representation has a formal mathematical basis in Predicate Calculus Theory.

The main feature of this type of knowledge representation is that built-in mechanisms associate rules as needed and can derive facts that are not explicitly stated in the database. For example, if rules state that construction costs associated with hilly terrain are moderate and that a certain area has hilly topography, the system can automatically infer that the construction costs for that area will be moderate.

The high level language Prolog efficiently implements predicate logic. Since this is the type of knowledge representation structure used in HLA, a more detailed description will be provided in subsequent chapters. An example of a hypothetical Prolog rule for determining whether the use of noise attenuation barriers should be considered at a certain location (*Point*) is shown below:

```
consider_noise_barriers(Point) :-
        technically_feasible(noise_barrier,Point),
        financially_feasible(noise_barrier,Point),
        high_noise_annoyance_levels(Point).

technically_feasible(noise_barrier,X) :-
        not accessibility_problems(Point),
        visual_disruption(noise_barrier,X,VisDsrupt),
        noise_attenuation(noise_barrier,X,NoiseAttn),
        VisDsrupt < NoiseAttn.

financially_feasible(noise_barrier,X) :-
        noise_barrier_cost(X,BarrCost),
        totalproject_cost(ProjCost),
        BarrCost / ProjCost < xx%.

high_noise_annoyance_levels(X) :-
        close_to(hospital,X)
        ; /* this means "or" */
        population_density(X,high),
        dBA_estimates(X,high).
```

1.4.3.2. Inference Networks or Semantic Nets

The nets are graphs formed by nodes that represent concepts, events or objects and links that specify functional and relational properties of the nodes. This formalism has been shown to be flexible and powerful as well as intuitively attractive from a psychological point of view.

During consultation, hypotheses or data are inserted into some of the nodes. Control mechanisms can then be used to propagate the information along the links, deriving conclusions and possibly detecting data inconsistencies.

Figure 4 represents a simple inference network that embodies the same knowledge described by the Prolog rule above.

1.4.3.3. Frames

Frames typically have a number of features that distinguish them from other representational systems: they are organized into hierarchies in which frames inherit information from their "ancestors", in an object-oriented type of environment. They have sub-units which can take on values or describe, in general terms, constraints on what these values can be. Frames are well supported by the high level language LISP.

1.4.3.4. State-Space Representation

This type of representation involves applying a set of operators to generate some or all the possible states following the current one. The resulting tree structure is then examined and an algorithm is applied to select the most desirable course of action. This scheme has been widely used by game-playing systems.

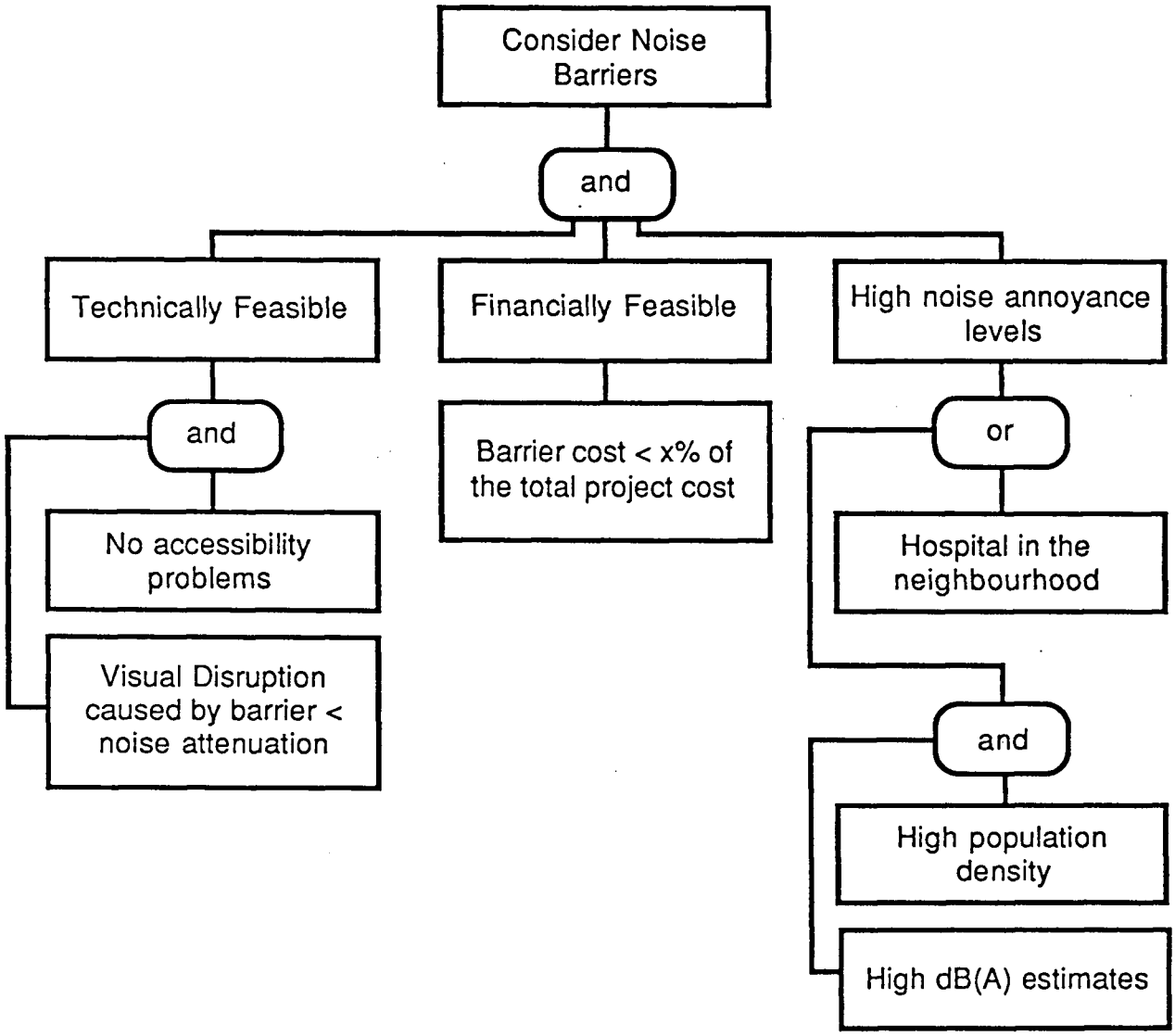


Figure 4. An Inference Network

1.5. Application to the Highway Corridor Location Problem

Corridor routing problems involve varying degrees of expertise in several different fields. The interdisciplinary approach currently used in major projects can be illustrated by the fact that the Kentucky Department of Transportation EAP (Environmental Action Planning) maintains an in-house staff with technical skills in about 40 disciplines [43].

There is no algorithm or formula which produces a highway corridor evaluation report automatically, based on the project data and characteristics of the region involved. There are, however, heuristic procedures that can aid the analyst in understanding the issues involved, and these can be supported by technical procedures. Pearse and Rosenbaum [66] describe a system under implementation that uses heuristics and default reasoning to evaluate the geological aspect of proposed road corridors.

Knowledge-based systems applied to the highway corridor location problem could aid the tasks of assessing impacts of alternative routes and justifying the choices made in a consistent way. The ability to make decisions and to justify them by displaying the rules used is considered much better than the more common "black-box" procedures, for it adds credibility to the choices made and allows for critiques and feed-back. In fact, explanation facilities are usually one of the most important

features of knowledge-based systems. Since the problems being addressed are usually ill-structured by nature and involve non-systematic procedures, it becomes essential to provide the user with a justification for the results obtained ("how?" questions) and how any information that is requested can help solve the problem ("why?" questions).

The spatial character of the corridor location problems constitutes one of the greatest difficulties related to this type of problem. Current knowledge-based techniques are time and storage intensive, and efficient processing of spatial data is one of the fields still in a research phase. For example, heuristic spatial search procedures have been developed. They save computational effort but provide no guarantee the solution obtained is the best possible. Conventional shortest path algorithms, on the other hand, yield the best possible solutions but are extremely time consuming.

The relative importance of different impact dimensions depend on the project's priorities and characteristics and on the concerns of the community. Trade-offs have to be established between pairs of possible dimension impacts, such as accessibility increases against noise and fumes pollution or construction costs against wildlife preservation. Utility theory describes procedures for evaluating these trade-offs [53]. HLA supports these coefficients but does not evaluate them.

Sets of rules were also developed to assess impact magnitudes. Examples of this type of knowledge include determining the potential for erosion based on slopes

and land cover or changes in construction costs according to slopes, soil types and presence of water bodies.

Finally, inexact reasoning mechanisms enable the user to use expected ranges of values or certainty coefficients as input data. The quality—in terms of precision—of the solutions obtained using the system reflects the precision of the information used in the analysis. This feature enables periodic solution refinements and is particularly important in engineering problems where the quantity of data tends to increase as time and money are spent during the execution of a project.

2. LITERATURE REVIEW

Two quite distinct topics were investigated in the review of the literature: knowledge-based systems and highway corridor location. While extensive, the literature on knowledge-based systems tends to describe successful implementations or is limited to the very general principles involved in the approach. This situation is partly due to the fact that expert systems are domain-specific by nature and partly because the techniques involved are relatively new. Highway corridor location, on the other hand, is an old problem. Many different methodologies have been developed and the literature on the subject is relatively abundant.

2.1. Knowledge-Based Systems

According to Sterling [33,34], building knowledge-based systems is currently an engineering skill. Experience in the last decade has led to a collection of working programs and an accumulation of accepted wisdom. This is found, for example, in the existence of commercial "expert-system shells". However, there seems to be no accepted methodology, and the literature tends to describe what is rather than prescribe what should be.

2.1.1. Categories of Knowledge-Based Systems

Stefik et al. [32] divided expert tasks into several categories and pointed out the fundamental requirements and key problems associated with each one.

One of the basic tasks described in that paper is ***planning***, defined as "creating plans that can be carried out to achieve goals without consuming excessive resources or violating constraints". If goals conflict, the planner has to establish priorities. If planning requirements or decision data are not fully known or change with time, the planner must be flexible and opportunistic. The main difficulties associated with planning systems are related to the size and complexity normally associated with planning tasks. If the details are overwhelming, the planner must focus on the most important considerations. In complex problems, there often are interactions between plans for different sub-goals. Uncertainty is usually present in this type of analysis and preparation for contingencies is required.

Another type of procedure described is ***design***. This involves specifying objects that satisfy particular requirements. The requirements are similar to those of planning tasks. One of the most serious obstacles that have to be faced in this case is that there is usually no comprehensive theory that integrates constraints with design choices. In addition, the system must be able to cope with interactions between sub-problems, record justifications for design choices and be able to explain its decisions later.

Design tasks must also address the problem of representing spatial data. Waterman [37] stresses the previously mentioned fact that excessive amounts of memory may be necessary to track spatial relations between groups of objects. Clever representation techniques are required to overcome the deficiency in this area.

2.1.2. Implementation Tools

The current popularity of rule-based systems for the solution of many types of practical problems is reflected on the number of implementation tools available for a variety of hardware environments. Wigan [39] lists over a hundred commercial software packages available for small systems, mostly "expert system shells" and LISP or Prolog compilers/interpreters. The shells are high level tools and do not offer great flexibility. For ambitious projects the capabilities of a full-featured language are recommended.

Many authors consider the Prolog language particularly well suited for expert system implementation [8,33,34]. The language contains its own backward-chaining inference mechanism and a powerful declarative language in which to express rule-based knowledge. It has the capability of dynamically asserting new rules generated by the system itself during a consultation. It is reasonably easy to write a Prolog meta-interpreter (that is, a Prolog interpreter written in Prolog itself) and then upgrade it to provide the features expected from the inference engine, such as explanation facilities and user query routines. Simple examples of such

meta-interpreters can be found in the literature [33,34].

The development of the knowledge base, including the choice of the knowledge representation scheme, is clearly the most difficult and time-consuming part of the task of developing an expert system. The inference engine, or rule-interpreter, can be especially designed or an off-the-shelf tool can be used. Commercial shells, however, tend to limit the flexibility with which knowledge can be expressed. The conclusion is that the inference engine should be tailored to fit the knowledge base, and not the opposite. Building one's own interpreter is not a time-consuming process, especially in contrast with the knowledge acquisition phase.

2.1.3. Inference and Explanation Capabilities

The basic reason for providing explanation facilities is to make the reasoning of the program transparent. This is extremely useful because it allows for discussion and analysis of the solution process. By analyzing the explanations provided by the system, an expert can suggest improvements to the knowledge base and inexperienced technicians can follow the reasoning of a seasoned practitioner.

Some types of explanations commonly mentioned in the literature are "how?", "why?" and "why not?", and the basic principle is always to keep a trace of the solution process and interpret it appropriately [33,34].

While expert shells normally provide explanations automatically, a number of techniques can be used to accomplish this task from high level languages.

Clark and McCabe [8] suggest adding an extra argument to each Prolog rule to keep the trace structure. Thus, adding an explanation facility to an existing system would mean changing every rule in the knowledge base.

A better approach for providing explanations is based on the previously discussed meta-interpreter approach and described by Sterling and Shapiro [33,34]. A standard meta-interpreter is modified to "remember" how a solution was reached by keeping a trace of the program execution. Once the proof is available, it can be interpreted and displayed as needed. This mechanism is extremely elegant and preserves the modularity of the system. Furthermore, the authors describe how it can be adapted to provide hierarchical "why" explanations.

2.1.4. Processing Uncertain Information

Knowledge-based systems usually have to deal with uncertainty. This is partly due to the fact that they often employ heuristic rules, inexact by nature, in the problem solving process. Some of the most important sources of uncertainty are described by Bonissone and Tong [5]. Uncertainty may arise from a lack of understanding of the processes involved in an analysis or from their intrinsically uncertain physical behaviour. Examples of the former are ill-defined concepts and weak implications in

the knowledge base, which occur when the expert or model builder is unable to establish a strong correlation between premise and conclusion. Another potential source of uncertainty is related to inaccuracies during data input or output.

Many researchers have worked in the field of uncertainty representation and propagation, and several approaches have been developed that allow systems to incorporate and handle uncertain information. The most familiar tools for dealing with uncertainty are the Bayesian operators (e.g. PROSPECTOR [17]), which use an effective likelihood ratio to quantify the strength of a given rule. This ratio measures the sufficiency of a given piece of evidence to prove a hypothesis.

One of the criticisms advanced concerning this approach is that the single value used to quantify uncertainty tells the system nothing about its precision. As far as the Bayesian operators are concerned, to say that the probability associated with an event is 0.5 might mean either 0.500 plus or minus 0.001 or 0.5 plus or minus 0.3, two very different pieces of information.

Another criticism against the expression of uncertain subjectivity with the use of Bayesian probability is that it cannot be used to deal with uncertainty in an efficient manner. In other words, Bayesian theory cannot distinguish between the lack of belief and disbelief, because it requires that *P(A occurring) + P(A not occurring) = 1*. A single value would combine evidence for and against the proposition without indicating how much there is of each.

The Certainty Factor approach (used in MYCIN [17]) is based on the Confirmation theory. It employs two separate values that indicate the belief and disbelief in the proposition. This approach is still subject to the criticism regarding precision, since both numbers are point values. It does, nevertheless, overcome the second objection mentioned. The system, however, suffers from a new disability. Since the two values are independent, they cannot be interpreted as probabilities and therefore there is no foundation of theory underpinning and justifying the interpretation and weighing for separate belief and disbelief measures [29].

A serious drawback of the Bayesian and Certainty Factor approaches is that they would generally require the complete joint probability distribution of all propositions. According to Quinlan [29], "... asking the system designer to specify separate values for each possible combination of evidence relevant to a proposition would overtax his knowledge and presumably his patience...". Some systems sidestep this problem by making conditional independence assumptions. These provide additional constraints that simplify the uncertainty propagation mechanism. In practice, however, these assumptions might not always hold and can lead to wrong and even absurd results.

Other types of logic have been proposed. Quinlan [29] described a system that does not assume variable independence, although it does account for it when specifically instructed. The conclusions obtained using this system, although generally less powerful than the ones given by the Bayesian operators, will always be correct (provided, of course, that the data and knowledge bases do not contain mistakes). An extra advantage of Quinlan's approach is that it allows for knowledge and data

consistency checks.

Some authors claim that the ideal approach would be to combine different uncertainty representation mechanisms and logics to fit the knowledge available [39].

The mathematical approaches have also been extended to fuzzy statements. Although fuzzy reasoning is not a rigorous approach, it provides an approximate reasoning process which is compatible with human intuition. The main advantage of fuzzy reasoning as an inference process is that it can yield plausible answers even in problems in which the conditions required for the mathematical approaches are not satisfied. Application of fuzzy reasoning generally involves the definition of membership functions and aggregation rules based on joining or intersecting fuzzy subsets. Aggregation rules can be defined for credibility and possibility, and methods that assess linguistic truth values have also been developed [18].

Highway location tasks involve the use of estimates and forecasts. The information available is frequently expressed in terms of tolerance intervals and is usually unknown (especially in early stages of a project) rather than intrinsically uncertain. Fuzzy logic is currently the most satisfactory for dealing with this type of uncertainty and was, therefore, selected for HLA.

Finally, it should be noted that, even though inference under uncertainty is currently one of the most popular topics among artificial intelligence researchers, consistent treatment of uncertainty has to date not been as important in practice as might

have been expected. The key problem in expert system applications is still the knowledge engineering rather than the finer details of the tool structure within which knowledge is articulated and delivered.

2.2. Highway Location Procedures

Many different approaches have been used to address the highway corridor location problem in the past. Most methods can be classified as pure judgement, economic analysis or rating schemes.

The pure judgement approach relies exclusively on the experience and intuition of an expert. Some sort of decision framework is always adopted for support and documentation purposes, but the quality of the final decision will depend on the level of expertise of the decision makers.

The economic analysis approach is widely used mainly because it provides a convenient, objective decision framework. However, it tends to underestimate the importance of attributes that cannot be measured in terms of currency. Cost-benefit studies represent the most typical example of this type of analysis.

Rating schemes represent a generalized form of economic analysis. Instead of using unit costs, however, attributes are weighted in terms of relative importance, or

preference. Some professionals feel uncomfortable with this approach, because it attaches numbers to attributes that are impossible to be formally measured. Another complaint about the method is that it aggregates all attributes in one final index that accounts for all the factors involved in the analysis but hides many relevant aspects of the solution such as the relative importance of each factor.

It has been advanced by researchers that short term memory can accommodate only seven pieces of information or concepts at once [59]. When more variables than this are present some are ignored by the decision makers or judgements become irrational. Decision support tools are necessary to overcome this human limitation.

In practice, corridor routing problems are often dealt with in an incremental fashion. A few major attributes are considered at first and lead to a number of almost equally acceptable options. Secondary factors are then included in the analysis as tie-breakers.

2.2.1. Collecting and Presenting the Data

Several procedures for evaluating environmental impacts have been developed over the last fifteen years, reflecting the increasing concern of society with factors such as natural resources and aesthetic values. Leopold et al. [54] present one such approach, which consists of an impact matrix where the magnitude and relative

importance of different impact dimensions are recorded. The purpose of this procedure is to enable reviewers of the project to focus their efforts upon the most significant aspects of the problem.

The procedure presented does not constitute a solution for the problem, but it can be used as a tool for assessing environmental impacts, which cover a significant number of impact dimensions.

McHarg [55] presented a method that involved the use of transparent overlays for each parameter used in the analysis. Each overlay represents one dimension of impact and is marked in three different tones, the darkest corresponding to the most serious physical and/or engineering obstructions to highway location. By overlaying all the parameters on the base map, it is presumed that the lighter areas will represent the ideal sites for the corridor, which has to be "eyeballed".

This method assumes all parameters have the same importance, which is not usually the case. In addition, the presented scale, from one to three, will in many cases be too restrictive. Nevertheless, it can be very useful for presentation purposes, as all the information available is condensed onto a single map. Clear conclusions, however, may or may not be easy to obtain and justify from this map. Furthermore, this method is rather time-consuming and expensive.

2.2.2. Multi-criteria Analyses

Multiple criteria decision problems have been discussed, among many others, by Massam [57,58] and Nijkamp [61,62,63,64]. Their approach involves the use of scaling techniques to represent the several dimensions of impact associated with each plan on a two-dimensional plane. The methods presented are quite general, but the interpretation of the results is rather subjective and in many cases it can be necessary to include reference cases—best and worst possible ratings for all criteria—to make the final result meaningful.

In these cases, these is no search for the best solution. The alternatives must be determined before applying the method and there is no guarantee that the best possible option is among the ones being considered.

2.2.3. Automatic Route Selection Systems

Several systems have been implemented that determine ideal locations for different types of corridors by dividing the study area into cells, assigning suitability indices to each cell and searching for the minimum negative (or maximum positive) impact path.

A general computer aided corridor location system (POWER) has been developed by

Giles et al. [50]. The system was originally intended for power transmission line corridors and expanded later into a general corridor location system.

One of the concepts discussed is the principle of equifinality: the greater the number of criteria that are taken into account, the more stable the final answer will be, since single errors become less significant and tend to cancel each other.

POWER assigns suitability scores to each cell according to a rating scheme and uses dynamic programming techniques to search for the highest social benefit (or least social impact) corridor.

Parameters can be added or removed from the analysis and, by changing coefficients in the objective function, one can estimate the stability of the solution given by the system. However, there is no way of directly assessing the precision of the solution.

POWER requires many inputs from experts. In the case where a corridor infringes endangered species habitats, or when time effects are relevant for impact assessment, external expert input, usually based on some set of heuristic rules, is necessary.

Dooley and Newkirk [45] also developed a corridor selection system that attempts to minimize environmental impacts. Their system incorporates a dedicated algorithm to assess composite impact values. Newkirk's cascade algorithm promotes multiple

lower level impact ratings to higher level impact ratings. It is argued that the algorithm yields better results than equal weight averages or threshold ratings; it is not intuitive, however, and does not provide much control over the process. Two parameters have to be arbitrarily set: the level below which ratings are not considered to be candidates for promotion, or "cascade level", and the number of identical ratings required for a single promotion to the next level, or "cascade base". The authors utilize a six-point integer scale for mapping purposes.

Another system, similar to POWER but specifically dedicated to highway corridor location, was developed by Civco, Kennard and Lefor [42]. The system was successfully used in the United States and found routes that had less adverse environmental impacts than those proposed by the Connecticut State Department of Transportation.

The system also uses a grid representation for the study area and includes a report module that summarizes numbers of cells impacted under different dimensions. Most of its characteristics are similar to those of POWER, including virtues and drawbacks. Being more domain-specific, however, it provides a slightly more comprehensive approach to the problem. It should be noted that the procedure described is not complete, as it accounts only for environmental impacts and disregards entirely social or economic factors.

The search mechanism used by the models described above consider only orthogonal moves between cells. The routes generated tend to have a squared-off

look, leaving an undesirable first impression as unrealistic and infeasible. Furthermore, the orthogonal grid representation has associated geometric distortions which can result in the generation of poor routes [52].

The systems reviewed were designed to perform calculations. They are useful and effective as engineering tools and represent a volume of documented knowledge that should not be discarded. However, conventional programs cannot cope with uncertain information or justify the solutions they provide. Therefore, it can be concluded that, in some instances, it would be desirable to incorporate conventional procedures into knowledge-based systems. These, in turn, could address aspects in which existing systems are weak, such as handling uncertain and qualitative information, and providing explanation facilities.

3. IMPLEMENTATION GOALS

Several objectives served as guidelines during the implementation stage of the Highway Location Assistant system. In order to achieve many of these goals, it was necessary to trade off the desirable attributes of computational speed and storage efficiency. In a prototype system, this does not represent a serious compromise. Major emphasis was naturally placed on those aspects of existing systems thought to be comparatively weak—flexibility and comprehensiveness. The following sections describe the basic goals of the project.

3.1. Flexibility

Flexibility is defined as the ease with which a system can be changed and expanded. It is a major requirement for expert systems, as the knowledge bases must be periodically analyzed, augmented and modified to improve performance and to incorporate, when available, new knowledge. Monolithic systems are fraught with interdependencies which can make them painfully difficult to modify. A modular design greatly enhances flexibility and makes the programs easier to document and maintain. Finally, since the interaction between different parts of the system is kept to a minimum, individual modules can be easily optimized or totally rewritten without interfering with each other.

3.1.1. Support for Several Levels of Operation

The system should have several levels of operation with respect to varying user knowledge and experience. A user familiar with the system and the principles of road corridor evaluation employed will need briefer explanations and less help on how to use the package. By contrast, a novice would have completely different requirements, perhaps trading off some power and efficiency for increased ease of use. Different levels of operation could avoid tedium or lack of comprehension, as the case may be.

Another advantage would be the ability to use the system at various levels of road corridor assessment. During the initial planning stage, standard analyses should be enough to identify initial possible alignments. In later stages, these potential routes can be evaluated in detail, providing more accurate comparisons.

3.1.2. Use of Uncertain Information

In many engineering projects, as time and money are expended, the quantity and quality of the data available tend to increase. Furthermore, much of the information processed in planning tasks consists of forecasts and has high associated uncertainty. For these reasons, engineers often prefer to express their beliefs in terms of lower and upper bounds or possible ranges of values instead of using single point values.

The obvious conclusion is that it would be extremely useful to have systems capable of processing uncertain information. Many approaches exist for this purpose (e.g. Bayesian operators, certainty factors, fuzzy logic). A problem with many of these is that they assume the information is intrinsically uncertain but well documented. If that is indeed the case, relatively complete and detailed conclusions can be obtained from the data using formal statistical procedures. However, if the necessary information is unknown, the system should not depend on it as an input.

In summary, an ideal system should be able to make the best possible use of the information available but not rely on data not easily obtained which may lead to detailed, but not necessarily correct, conclusions.

3.1.3. Search and Evaluation Procedures

The process of highway route location usually involves evaluating alignments proposed by different groups or agencies as well as independently searching for the best possible route. Consequently, it would be desirable to have both route evaluation and search capabilities as parts of the system.

The search routines should be flexible enough to allow easy refinements of the solution and benefit from all the information the user can provide. For instance, it should be possible to define search areas, ruling out regions that are obviously unsuitable for a highway. The user should be able to begin the analysis with a

large search area and low search resolution and refine the solution incrementally. Non-orthogonal moves should be allowed to reduce geometric distortions associated with typical squared-off solutions.

Finally, the objective function should be easy to modify in order to allow different criteria to be used in the search.

3.1.4. Representation of Spatial Data

The spatial character of the problem can present a serious burden in terms of storage requirements and computational speed. Conventional programs usually divide the study area into cells and use large matrices to store characteristics of each cell. This approach has several serious drawbacks.

The data stored are highly redundant, since regional attributes (topography, geology and so on) tend to be repeated across many cells.

As conventional matrices are rectangular, irregularly shaped study areas cause further inefficiencies in memory utilization. Although this problem could be alleviated through use of sparse matrix techniques, this would involve additional computational overhead.

However, the most serious problem with this approach is its lack of flexibility. For

example, to refine preliminary solutions, it is necessary to re-tabulate all the data into a finer grid.

Ideal data structures should combine time and storage efficiency, support areas of any shape and allow for easy data entry, corrections and refinements.

3.2. Comprehensiveness

In situations when the required data are not available, the system may have to use default values and make assumptions which will generally have higher associated uncertainty than user-provided information. In such instances, the user should be provided with explicit information about any assumptions.

Selective explanation facilities are paramount. Exhaustive explanations which produce screenfuls of text are not very helpful if the user is actually interested in one specific aspect of the solution. Effective explanation facilities enable the user to control which parts of an explanation she wants displayed, browse through intuitive sections of the solution, concentrate on the less clear parts and skip others altogether.

3.3. Accessibility

One of the factors which determine the usefulness of any system is its availability. This is especially true for expert systems, as one of their major advantages is the availability of their expertise and knowledge on a twenty-four hour basis, in dispersed geographical areas. This is one reason why it was decided that HLA should be implemented on a microcomputer. Commonly, most professionals have access to these machines and feel comfortable with the highly interactive environments they can provide.

Still with the goal of keeping the system open, easy to understand and update, it was decided that the main implementation objectives should be code clarity and conciseness rather than computational efficiency. In practice, this implied using a relatively popular fifth generation language, interpreted Prolog, to write the whole system. However, it would be possible to significantly improve computational performance in future stages by recoding some of the key routines in a lower level language such as C or assembly.

4. DESCRIPTION OF THE SYSTEM

HLA was implemented on an IBM personal computer equipped with 640 kilobytes of RAM, two 10 megabyte hard disks and an 8087 math co-processor. The system, presently consisting of approximately 50 kilobytes (about 30 pages) of source code, will run on standard IBM PC or compatible microcomputers with a minimum 384 kilobytes of RAM and a hard disk.

The system is logically divided into modules. For illustration purposes, the modules and their current relative sizes are depicted in Figure 5. Note that most modules tend to be fairly static in terms of size, except for the knowledge base that would tend to grow significantly as the system becomes mature. It is estimated that the relative code size of the Knowledge Base module, presently 45%, will grow to about 95% as the system evolves into production stage.

The Knowledge Base module contains all the evaluation predicates. These consist of facts and rules that describe how to assess impacts associated with a variety of entities such as subareas, road segments or complete highway alignments. It is the most domain-specific module.

The predicates in the Inference Engine module control program execution and provide explanation facilities and most of the input and output.

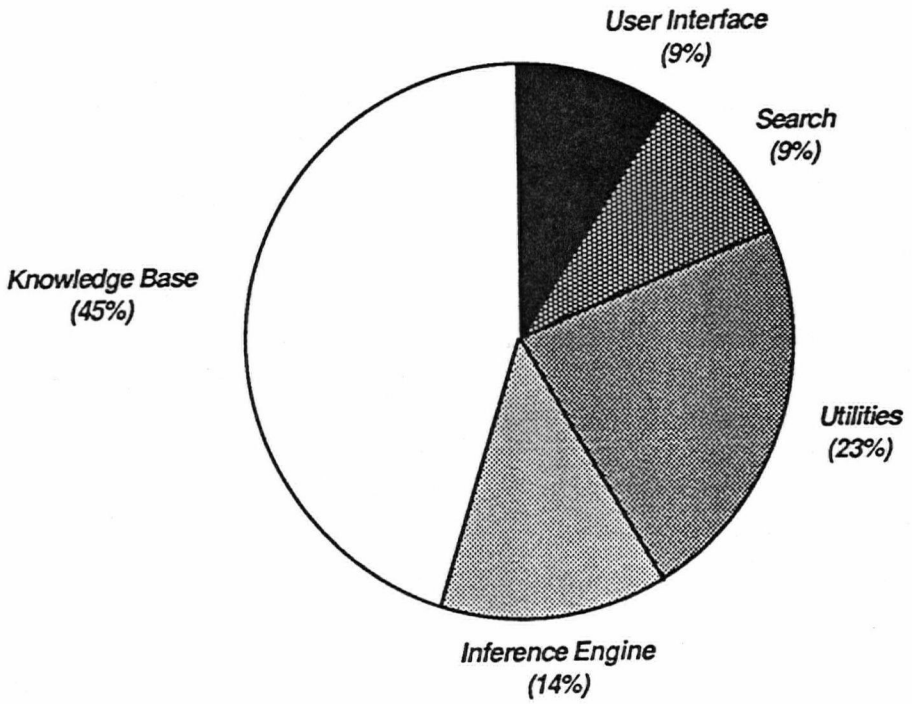


Figure 5. The Highway Location Assistant Components and their Sizes

The User Interface module acts as a program driver. It hides the lower-level predicates from the user and provides simple predicates that generate sequences of commands, making the system relatively user-friendly. It enables experienced users to define their own interfaces or bypass this module entirely, gaining access to the whole system. Input and output graphic capabilities can also be incorporated into the User Interface module. Figure 6 shows how the predicate ***segment*** displays the fuzzy impacts associated with a given route segment.

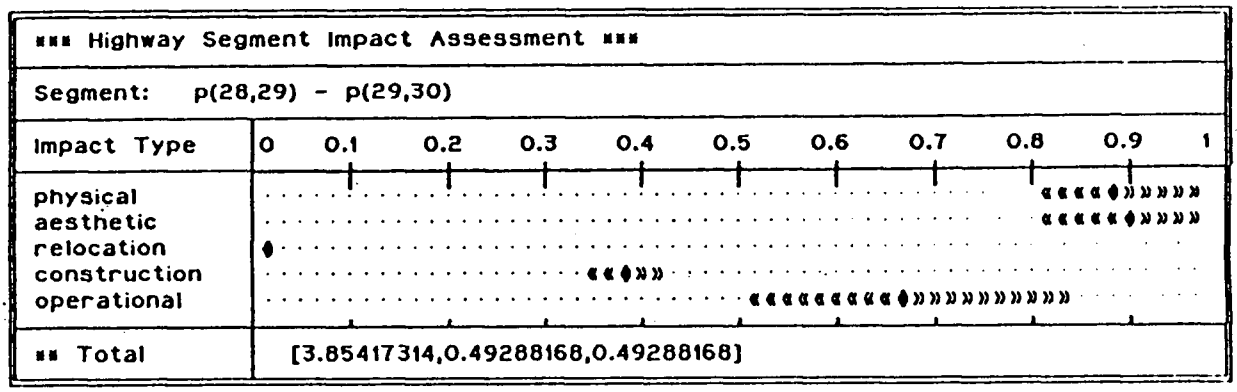


Figure 6. Sample Point Evaluation Graphical Output

Finally, the Utilities module contains predicates related to geometry and fuzzy numbers as well as other general-use procedures.

The modular design allows several levels of operation. Inexperienced users can use less flexible predicates through the user interface module, while users who are familiar with the system can call more primitive routines or even bypass the explanation facilities and inference engine module (and use Prolog's built-in inference mechanism).

4.1. Implementation Tools

HLA is a relatively complex system, having several highly specialized modules and

data structures. For this reason (in addition to the ones specified in previous sections) Prolog was selected to implement HLA rather than one of the less flexible expert system shells.

The task of designing a Prolog program does not involve specifying an algorithm in the same way as in a conventional programming language. The Prolog approach is to describe known facts and relationships about a problem as opposed to prescribing the sequence of steps to be taken by the computer to solve the problem. To a large extent, the language itself then determines how the information available can be used to reach acceptable solutions. Thus, Prolog can be viewed as a descriptive language as well as a prescriptive one.

Prolog is a practical and efficient implementation of many aspects of so-called "intelligent" program execution, such as non-determinism, parallelism and pattern-directed procedure call. A uniform data structure, called the ***term***, is used to represent all data and program statements. The fact that program statements can be viewed as data is the cornerstone of HLA's inference engine, explanation and user query facilities.

To understand HLA's operation and the code samples included in this chapter, it is necessary to be familiar with the basic Prolog syntax and with the mechanisms it employs to accomplish its tasks. The *de facto* standard Prolog features are described in detail by Clocksin and Mellish [9], and some of the key aspects are outlined below.

In Prolog, variable identifiers begin with uppercase characters or underscore (_) and literal constants (***atoms***, in the Prolog jargon) with lowercase. Variables are free (or ***uninstantiated***) when they have no value assigned, as opposed to ***instantiated*** ones which can be bound to constants, to other variables or to partially bound structures that contain other variables. ***Clauses*** can have a body (the ***IF*** portion) and a head (the ***THEN*** portion). The bodies can contain conjunctions (***AND*** conditions), disjunctions (***OR*** conditions) or nothing, in which case the clauses are called ***facts*** and assumed to be always true. In addition, since Prolog has a dual—prescriptive and descriptive—nature, some flow control structures are provided (e.g. ***cuts***).

Prolog performs tasks in response to questions posed by the user. A question provides a conjunction of goals to be satisfied. Facts can cause goals to be satisfied immediately, whereas rules have the effect of reducing the task to that of satisfying a conjunction of sub-goals. However, only clauses that match the goal under consideration can be used. The matching process (also called ***unification***) corresponds to parameter passing in traditional languages.

When a goal cannot be satisfied, a backtracking process takes place, whereby Prolog tries to re-satisfy previous goals by finding alternate ways of solving them. Additionally, the backtracking process can be explicitly requested to provide all the possible answers to a given question.

4.2. Capabilities

HLA has predicates that support several types of analyses. All of them have common characteristics such as the ability to use fuzzy information, explanation facilities and default reasoning. The two evaluation predicates (***evaluate_point*** and ***evaluate_segment***) are strictly rule-based. They rely on clauses in the knowledge-base to emulate the steps an expert would take to assess the suitability of an area for a highway. The other predicates, while taking advantage of the fuzzy operators and of Prolog's dynamic data structures, implement essentially conventional procedures. Regardless of their nature, all predicates are totally integrated.

The basic predicate ***evaluate_point*** is the building block on which the more complex procedures rely. The call ***evaluate_point(P,L,I)*** takes a point as input and returns a scaled list of impacts (with their types and magnitudes) and a global impact index that can be used to compare the suitability of different points.

The next step involves evaluation of a segment of corridor, given a certain length and central point. It takes into account interactions with linear entities such as existing highways, cattle and wildlife crossings.

Using the above predicates, a whole route can be analyzed given an open polygon and step size. Finally, optimal routes can be generated according to specified criteria

(minimum uncertainty, minimum expected impact, minimum possible impact etc.) given the starting and ending points, a search area and a step size. The hierarchical predicate structure is depicted in Figure 7.

Experienced users can also call lower level predicates to estimate particular impacts such as construction costs or visual impacts associated with a particular route.

4.3. Data Representation

HLA's database is composed of two types of information: global and regional data. Data that apply to a whole project (e.g. expected traffic levels and traffic mix, number of lanes, project life etc.) are global. Regional data are used to describe portions of the study area.

While global data can be easily represented by use of project constants, regional data has a spatial character and usually presents a serious burden in terms of storage requirements and computational speed.

4.3.1. The Basic Data Structures

An original data representation technique is employed by HLA to efficiently store and access regional data, taking advantage of one of Prolog's strengths—list

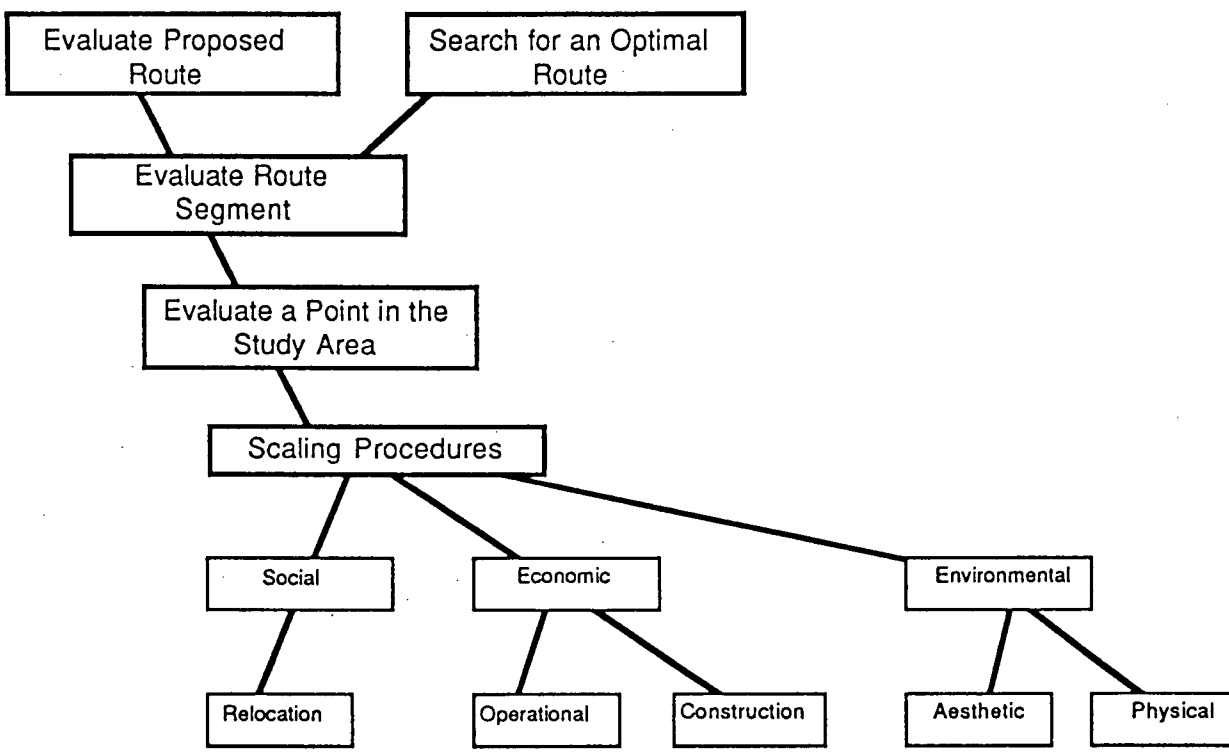


Figure 7. The Highway Location Assistant's Predicate Hierarchy

manipulation. The study area is divided in homogeneous zones, or subareas, which are defined as polygons. Each subarea is represented by a structure (of ***area*** type) that has two members. One is a list of vertices that define the polygon and the other (also a list) contains characteristics of the subarea such as topography, geology and land use. Several maps can be overlaid to describe all the significant aspects of a given study area. Linear spatial entities such as existing highways or cattle crossings are represented by similar structures where the first list contains an open polygon. An example of how the structures described above are used can be found in Chapter 5. Appendix 1 contains a sample data file that describes the regions depicted in Figures 16, 17 and 18.

The method described is extremely economical in terms of memory requirements, making it possible to analyze extensive and highly detailed areas with a microcomputer. Furthermore, this approach makes data entry and editing much easier than with the conventional matrix-based method.

4.3.2. Extracting Information from the Data Structures

To analyze a certain point, information must be extracted from the data structures described above. This involves increased computational effort (in comparison with the traditional techniques) but enables most of the information to be kept in memory, minimizing relatively slow disk access. Overall, there is a significant gain in speed.

Primitive geometry-related predicates were defined to extract information from the structures described above. The simplest one (***distance(P1,P2,Distance)***) gives the distance between two points. It is used to analyze impacts relative to entities that can be represented by points, such as hospitals and schools.

To determine interactions with linear entities such as existing highways or cattle crossings, another predicate was defined. The clause ***intercept(P1,P2,P3,P4)*** succeeds if the straight line segment from P1 to P2 intercepts the one from P3 to P4.

One of the most interesting predicates in this group is the one used to retrieve information from the ***area*** structures: ***in_area(Point,Area)*** succeeds if the polygon ***Area*** contains ***Point***.

A straightforward solution to this problem is presented by Sedgewick [69]. If a long line segment is drawn from ***Point*** in any direction (long enough so that its other endpoint is guaranteed to be outside the polygon) and the number of lines from the polygon it crosses is ***n***, then the point must lie inside the area if ***n*** is odd, and outside otherwise. This can be easily seen by tracing what happens as the line segment is traversed from the outside point. After the first side of the polygon is hit, the segment is inside the polygon; after the second, it is outside and so on. The algorithm used by HLA is based on the same principle:

- Create a list of the relative position of the point with respect to each side of the polygon. All possible situations are illustrated in Figure 8.
- The point lies inside the area if there is an odd number of sides to the left, right, above and below it (or, of course, if it lies on at least one of the segments).

If the language being used supports lists, this algorithm is slightly more efficient than the one suggested by Sedgewick, because fewer operations are usually required to determine the relative positions of points with respect to line segments than to verify whether two line segments cross. For illustration purposes, Figure 9 depicts

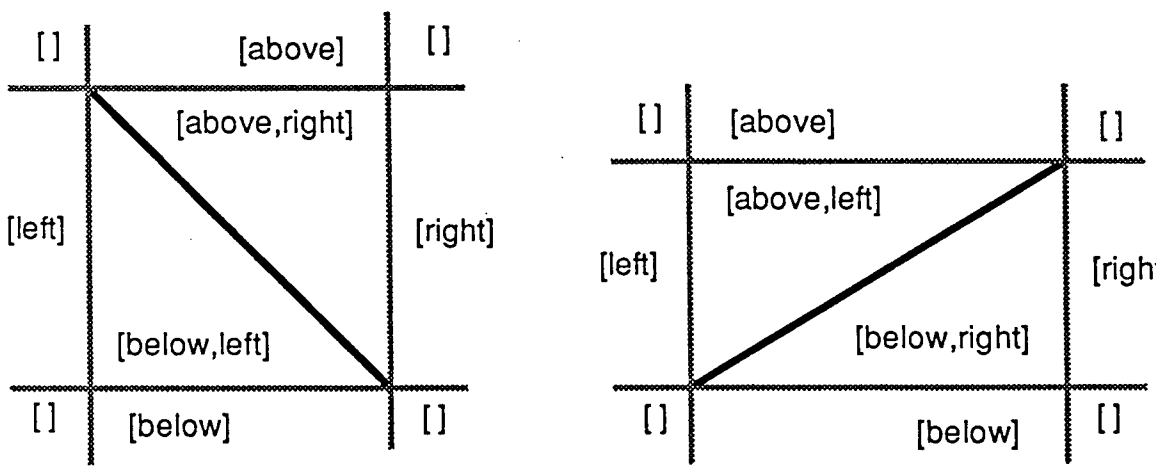


Figure 8. Position of a Point with Respect to a Line Segment

several points and their corresponding lists of relative positions with respect to the sides of a sample area.

To demonstrate how this predicate is used in the system, the definition of the predicate ***region_data*** is given below:

```
region_data(Point,Data) :-
        area(Area,Attributes),
        member(Data,Attributes),
        in_area(Point,Area), !.
```

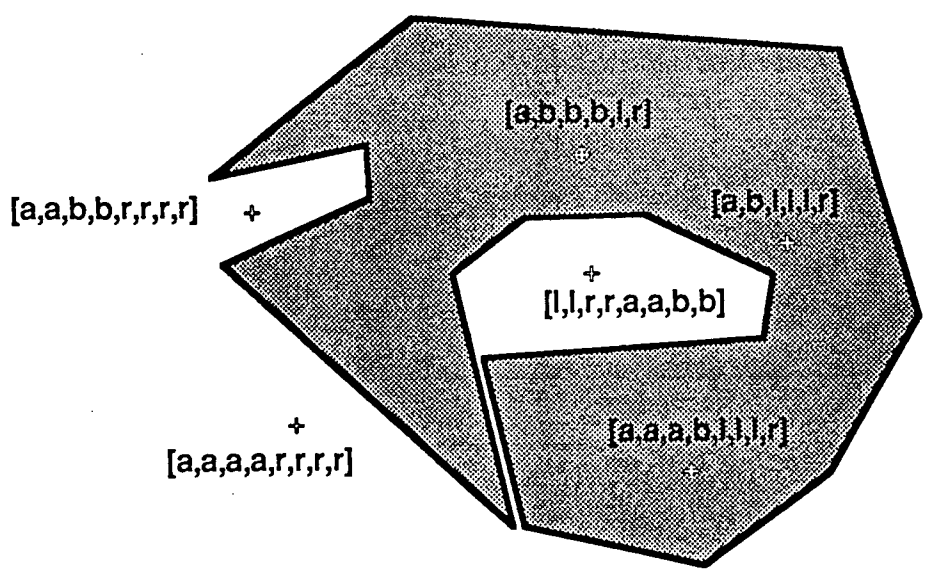


Figure 9. Sample Use of the IN_AREA Predicate

When this predicate is called, ***Point*** should be instantiated, that is, it should contain a value. ***Data*** can also be instantiated, in which case the predicate will succeed if the data actually corresponds to the point given. Usually, however, ***Data*** will be partially instantiated and the predicate will succeed and return the desired value.

For example, to satisfy the goal ***region_data(p(2,2),topography(T))***, where ***Data*** is partially instantiated to ***topography(T)*** and ***T*** is variable, Prolog would:

- Select the first ***area*** structure in the database.
- If possible, unify ***topography(T)*** with a member of the attribute list of the area, automatically instantiating ***T***. If this is not possible, the next area is selected.
- Test whether the point ***p(2,2)*** lies inside the selected area. If it does not, try the next ***area*** structure. If there are none left, fail.

This relatively simple predicate is declarative by nature. From a procedural viewpoint, however, it appears fairly complex. The definition relies on Prolog's unification mechanism and non-determinism.

A subtle point should be made about the ***region_data*** predicate. The test that is easier to perform, ***member...*** is applied before the more time-consuming ***in_area***. If the desired information is not in the attribute list, there is no need to check whether the point lies inside the area. This type of optimization, a practical application of the motto "look before you leap", is highly desirable since it improves efficiency without sacrificing code readability.

The definition of the ***region_data*** predicate also enables the user to define superimposed areas. In this case, the first areas to be defined have the precedence and are subtracted from others subsequently declared. This feature is particularly useful to describe regions that have a fairly uniform background or areas which contain others. The technique is demonstrated in chapter 5, a sample run of the system.

4.4. Evaluation Predicates

The evaluation procedure employed by HLA is a form of a rating scheme. To address the concern that this type of method aggregates all attributes in one final index that hides many of the factors involved in the analysis, the evaluation routines also return a scaled list of the impacts being considered, their type and extent. This makes it possible to examine the significance of each type of impact in the aggregated index.

The extent of each type of impact is evaluated independently through use of rules that can reflect conventional procedures, heuristic methods, or combinations of both. These values are then scaled to make them homogeneous and enable comparisons. The next section describes this process in detail. Since all values provided by the higher level predicates are scaled, they can be used for comparison purposes only. However, lower level predicates can be used to directly assess individual variables

such as construction or operating costs. Several steps are involved in the evaluation of each corridor segment, as discussed in the following sections. Figure 7 illustrates HLA's conceptual framework for route evaluation and search.

4.4.1. Evaluation of the Individual Impacts

The system utilizes sets of rules for each type of impact assessment. The process can mimic the procedures employed by conventional systems and combine them with heuristic approaches to obtain impact estimates. Currently, HLA has a knowledge base which consists of about one hundred and fifty rules and facts for impact evaluation. This number is adequate for a prototype system. However, it is estimated that in order to achieve expert level of performance, the number of rules in the knowledge base would have to be expanded by a factor of ten to twenty times.

The rules presently implemented allow for the assessment of several types of impacts under three major categories; environmental, social and economic impacts.

Environmental impacts are logically divided into physical (noise, air and water quality impacts) and aesthetic attributes (visual impacts). All environmental impacts are assumed to be proportional to the population density of the subarea.

Noise impact, in addition to the number of people affected, is assumed to be

proportional to the *Leq* (mean noise level over a twenty-four hour period, expressed in decibels) resulting from vehicle traffic. Air quality impacts are measured in terms of vehicle emission levels and characteristics of the land cover. Finally, distance to waterbodies, accident probability and the expected volume of trucks with hazardous loads are used to estimate possible water quality impacts.

Social attributes are presently measured in terms of necessary relocation of residences only. This is assumed to be directly proportional to the population density in the area.

Finally, the economic impacts considered comprise construction and operational costs. Land acquisition, earthwork and foundation costs are based upon region attributes such as topography, land use and geology, supplemented by the cost of special structures such as interchanges and cattle underpasses. Operational costs are composed of vehicle operation (which in turn depends on expected traffic mix and volume, grades and design speed), travel time monetary costs and accident costs.

4.4.2. First Scaling Procedure

To enable comparisons between the contributions of each particular type of impact to the total aggregated index, all the values obtained are scaled from 0.0 (best) to 1.0 (worst). To accomplish the scaling, the system keeps track of the maximum and minimum possible values for each type of impact. For example, if the noise levels

range from 40 to 80 dB(A) and visual impacts are rated in a scale from 1 to 7, a value of 60 dB(A) and a visual impact index of 4 would both be rated as impacts with 0.5 magnitude. The underlying assumption is that impact scales are linear. This does not constitute a serious limitation, since it is relatively easy to transform non-linear scales (e.g., to "linearize" the noise level scale, decibels are defined in terms of the logarithm of the energy transmitted by the noise source).

If the knowledge base is modified (as it is likely to be in any expert system) or a combination of factors that never happened before occurs, the limit values for each impact could occasionally have to be readjusted. To enhance modularity, reduce the potential for errors and ease the tasks of the knowledge engineer, HLA automatically fixes these limits whenever they are exceeded. It should be noted that when this mechanism is applied HLA effectively modifies itself to remember the new extreme values, so results obtained before and after the adjustments cannot be directly compared. Obviously, this might at times cause some inconvenience. These, however, are far outweighed by the aforementioned benefits that such a mechanism represents.

4.4.3. Impact Aggregation

An aggregated impact estimate is also provided by HLA's evaluation predicates. This index summarizes the individual impacts, reflecting the fact that some types of impacts can be considered more important than others by the community.

To obtain this global index, simplified utility functions are used to evaluate the trade-offs among different attributes. Utility functions can have different forms, the simplest one being the Additivity Value function. This type of function can be shown to represent the underlying preference of a decision maker provided each pair of attributes is preferentially independent of their respective complements [53]. The form of the Additivity Value function is:

$$v(x_1,x_2,\ldots,x_n) = v_1.x_1 + v_2.x_2 + \ldots + v_n.x_n$$

where v is the utility associated with the vector $\{x_1,x_2,\ldots,x_n\}$,
x_i is the level of the attribute i and
v_i are component value functions.

The component value functions represent the decision maker's preferences: for each level of a certain attribute, they return a scaled utility value.

To avoid the difficulties of assessing meaningful component value functions and to remain consistent with the current practice (when this type of approach is adopted), the general procedure was simplified by assuming all the component value functions to be constant values. These values can then be interpreted as relative weights of each attribute which should reflect the preferences of the communities involved.

HLA has built-in defaults for the attribute weights which can be used in preliminary phases of a project. However, it should be noted that these values represent strictly individual values and preferences. To establish these values, a system would

have to have not only technical expertise but also the capability to make decisions in social and political fields.

4.5. Inference and Explanation Facilities

The typical decomposition of expert systems into knowledge base, inference engine and database can become blurred for systems written in Prolog, the reason being that the language itself provides much of an inference engine. However, Prolog does not directly provide features expected of expert systems that are usually embedded in the inference engine. Examples are generation of explanations and user query facilities.

To provide these features, an enhanced Prolog meta-interpreter was added to HLA. A meta-interpreter, as explained in section 2.1.3 (Inference and Explanation Capabilities), interprets programs written in its own language. Simple examples of such programs are described by Sterling and Shapiro [33,34]. The sections below present a step-by-step description of HLA's meta-interpreter:

4.5.1. A Trivial Meta-Interpreter

An extremely simple Prolog meta-interpreter can be written as:

```
solve(true).
solve((Goal1,Rest)) :- solve(Goal1),solve(Rest).
solve(Goal) :- clause(Goal,Body),solve(Body).
```

The first clause is trivial. What it actually says is "if the goal is true, then solve has succeeded". Facts from the database will satisfy this clause.

The second clause simply breaks up a compound goal (a conjunction) into two parts: the first goal and the others. Note that this is accomplished directly, in the head of the clause. The next step is to solve each part, which is done via recursive calls to the ***solve*** predicate itself. This clause will only be used when the argument is a conjunction.

Finally, if a non-compound goal is not a fact, it may be the head of a clause in the program. The standard predicate ***clause*** is used to retrieve the body of the clause (it could be a fact, a conjunction or the head of another clause) and ***solve*** is called once more.

When any of the clauses succeeds, Prolog places a marker on that point of the execution. Subsequent failures will then cause it to backtrack and try a different solution from that point on, effectively covering the whole solution tree. This mechanism not only ensures that the final answer will be obtained (if there is one), but also generates all the possible alternate solutions to a given query.

This example serves only for illustration purposes, since it does not incorporate any enhancements to the original language (in fact, it does not even support the full standard Prolog syntax. HLA's corresponding code consists of eight clauses). It does, however, provide the basic means to monitor the built-in inference engine and to modify it.

4.5.2. Adding User Query Capabilities

User query capabilities can easily be added to the basic meta-interpreter by adding the extra clause:

```
solve(Goal) :-
        askable(Goal),
        ask(Goal,Answer),
        respond(Goal,Answer).
```

The first sub-goal assumes that a procedure ***askable*** is defined, which specifies when goals that the interpreter fails to prove by itself can be delegated to the user. For example, ***askable(adt(X))*** indicates that the system can ask the user what the average daily traffic will be in case this value cannot be found in the database.

The second sub-goal will actually query the user and read the response (into the variable ***Answer***).

Finally, ***respond*** will assert the answer into the database to avoid asking redundant questions in case the value is needed again.

4.5.3. Adding "WHY?" Explanations

The next logical improvement allows the interaction to go the other way: when asked a question, the user can answer with another question.

Answers to "why?" questions can be useful when the systems asks for values whose use is not obvious to the user. To provide this facility, an extra argument is added to the ***solve*** predicate described above. The extra predicate contains a list of the rule being used and its ancestors. They must be explicitly represented by an extra argument since there is no access to the global state of the computation in Prolog programs. The new ***solve*** predicate is listed below:

```
solve(true,true).
solve((Goal1,Rest),Rules) :-
solve(Goal1,Rules),solve(Rest,Rules).
solve(Goal,Rules) :-
        clause(Goal,Body),
        solve(Body,[rule(Goal,Body)|Rules]).
solve(Goal,Rules) :-
        askable(Goal),
        ask(Goal,Answer),
        respond(Goal,Answer,Rules).
```

With a list of the rules being used, "why?" questions can be immediately implemented by adding the following clauses to the ***respond*** predicate:

```
respond(why,Goal,[Rule1|Rest]) :-
        display_rule(Rule),
        ask(Goal,Answer),
        respond(Answer,Goal,Rest).
respond(why,Goal,[]) :-
        print(" No further explanation possible "),
        ask(Goal,Answer),
        respond(Answer,Goal,[]).
```

When a "why?" query is entered, the first clause displays the current rule and discards it by recursively calling ***respond*** with the ***Rest*** of the rules as the third argument. If another "why?" query is entered, the parent rule will be displayed, then the grandparent and so on until the list is empty. The second predicate then simply displays the message "No further explanation possible".

Of course, extra ***respond*** clauses are necessary to accept actual values provided by the user. Additionally, HLA accepts ***unknown*** as an answer. The corresponding clause is

```
respond(unknown,Goal,_) :-
        default(Goal),
        assert(Goal),
        print(" In this case I'll assume ",Goal).
```

This clause assumes there is a default value for ***Goal*** in the database (typically with high associated uncertainty). The default value is asserted into the database as a fact and the user is informed of the assumption. Examples of system-user interaction during an HLA run can be found in Chapter 5.

4.5.4. Adding "HOW?" Explanations

The principle behind "how?" explanations is identical to that used to answer "why?" queries: a trace of the execution is kept and displayed upon request. However, while "why?" queries involved a single, local chain of reasoning, answers to "how?" queries require that the complete solution be kept and displayed. Figure 10 shows the difference graphically. Each Prolog rule can have any number of children rules, but one and only one parent.

HLA implements a ***prove*** predicate that is similar to ***solve*** but stores a tree structure to represent the whole proof instead of a list of ancestor rules. The ***prove*** predicate is complemented by ***interpret***, which displays the proof in readable form.

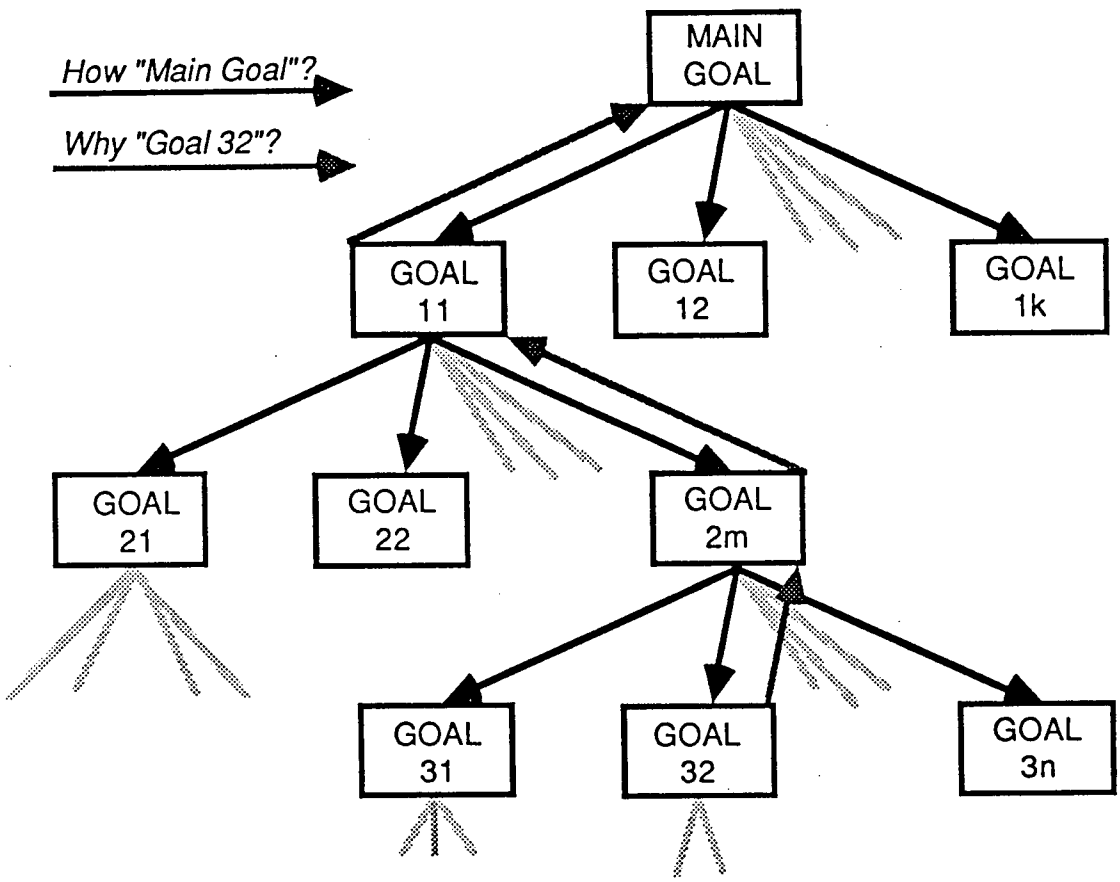


Figure 10. A Goal Tree with WHY and HOW traces

Appendix 3 shows part of a "how" explanation. The user can select portions of the explanation of interest. This avoids exhaustive explanations that can take considerable amounts of time and become rather confusing. This feature is implemented by the following clause:

```
interpret(implies(Proof,Goal,Body)) :-
        print(Goal),
        want_to_go_on,
        display_rule(rule(goal,Body)),
        interpret(Proof).
```

After the goal is displayed, ***want_to_go_on*** asks if the user is interested in the proof for that sub-goal. If the answer is ***yes***, the rule is displayed and ***interpret*** is called to go on with the explanation. Otherwise, the whole clause fails, the children branches of the proof tree are skipped and the next clause of the ***interpret*** predicate carries on with the explanation of the next sibling branch.

4.5.5. Improvements to the Basic Mechanism

The explanations given by the procedures described mirror exactly the Prolog computation, which may not be what is desired in certain cases. Using meta-interpreters allows greater flexibility, for explanations can be given which differ from the logic of the program itself yet provide the justification for which the rule

was derived.

This type of situation arises when explaining fuzzy operators, where rather than showing the sequence of operations (which are of little use to most users) a reference is displayed to the paper that describes the extension principle and derives the operators. Additionally, it may not be desirable to keep the whole proof tree when the geometry related predicates are used. Concepts such as points contained in areas or intersecting lines are obvious to users but involve the use of many rules, which could compromise the system's performance without bringing any real benefit.

The clauses that implement explanations not directly based on a trace of the execution are:

```
prove(Goal,implies(Proof,Goal,explain)) :-
        explain(Goal,_),
        call(Goal).
interpret(implies(true,Goal,explain)) :-
        print(Goal),
        want_to_go_on,
        explain(Goal,Explanation),
        print(Goal," ==>  ",Explanation).
```

The first clause checks whether there is an explanation available in the knowledge base. If successful, it stores the token ***explain*** in the proof tree and uses the

standard ***call*** predicate to solve the goal without keeping track of the partial solution. Note that it does not store the explanation in the proof tree.

The second clause detects the token in the tree, retrieves ***Explanation*** from the knowledge base and displays it to the user.

This feature not only enhances the clarity of the explanations but also improves efficiency in two ways. Space is saved by keeping only a token in the tree rather than the rest of the actual solution and speed gains are achieved when ***call*** passes control of the execution to Prolog's native inference engine.

Another weakness of the simple meta-interpreter described above is the inability to handle the Prolog's standard procedure ***findall***. This is a handy predicate that allows multiple solutions of a problem to be collected in a list. Unfortunately, its mechanism differs slightly from other standard predicates and the interpreters described so far cannot trace its execution (nor can Prolog's built-in trace facility). Since HLA uses ***findall*** to form lists of impacts, an extra clause had to be developed to make its execution transparent:

```
interpret(implies(system,findall(_,Goal,_)) :-
        prove(Goal,Proof),
        interpret(Proof),
        fail; /* Note that ";" means "OR" */
        true.
```

The head of the clause ensures that it is used only when the token ***findall*** is encountered in the proof tree. The first two sub-goals prove the goal once and interpret the proof. The ***fail*** predicate never succeeds and is used to force Prolog to backtrack and try to solve the goal as many times as possible. When all the solutions have been found, the first part of the disjunction will always fail and the second succeed.

The clauses ***interpret*** and ***prove*** call each other and themselves. It is interesting to note that this example of mutual and self recursiveness is relatively easy to understand from a declarative viewpoint. If viewed procedurally, however, it is extremely difficult to follow the actual program flow involved except for the simplest examples.

In addition to the features described in this section, HLA's inference engine incorporates several other enhancements to the meta-interpreters found in the literature. These include the ability to handle disjunctions and program flow control structures (***cuts*** and ***snips***) and a predicate that displays rules with compound bodies in a clear way, using indented blocks.

4.6. Search Procedure

HLA can be used not only to evaluate suggested routes but also to locate optimal

alignments according to an objective function specified by the user. The mathematical procedure employed is based on Dijkstra's shortest path algorithm [44,50]. Some of the common criticisms advanced against this type of procedure are also addressed, particularly the squared-off look of the routes generated and the geometric distortions associated with this type of approach. The enhancements to the basic procedure are described below, in section 4.6.2.

4.6.1. Classical Aspects of the HLA Implementation

To apply the modified Dijkstra algorithm, a virtual grid is superimposed onto the study area. The links are actually road segments, dynamically generated by the procedure. To locate the path that minimizes an objective function ***Z***, the following sequence of steps is applied:

- Create a dummy link with associated impact 0.0 that connects the destination point to itself.
- Expand the graph by adding links that originate from nodes that are part of the graph and end in nodes that are not (but lie within the search area). These links are called "candidates".
- Select the candidates with least cumulative ***Z*** value and add them to the graph.
- If the origin is already part of the graph, stop, otherwise repeat from step 2 (expand the graph further).

When this phase is completed, the graph will have a tree configuration with at

least one branch linking the origin and the destination. Figure 11 shows how the graph is generated.

To determine the branch that represents the optimal path, the tree is traversed from the origin to the destination. The second phase of the procedure is much simpler than the first; it is accomplished by the following Prolog predicate:

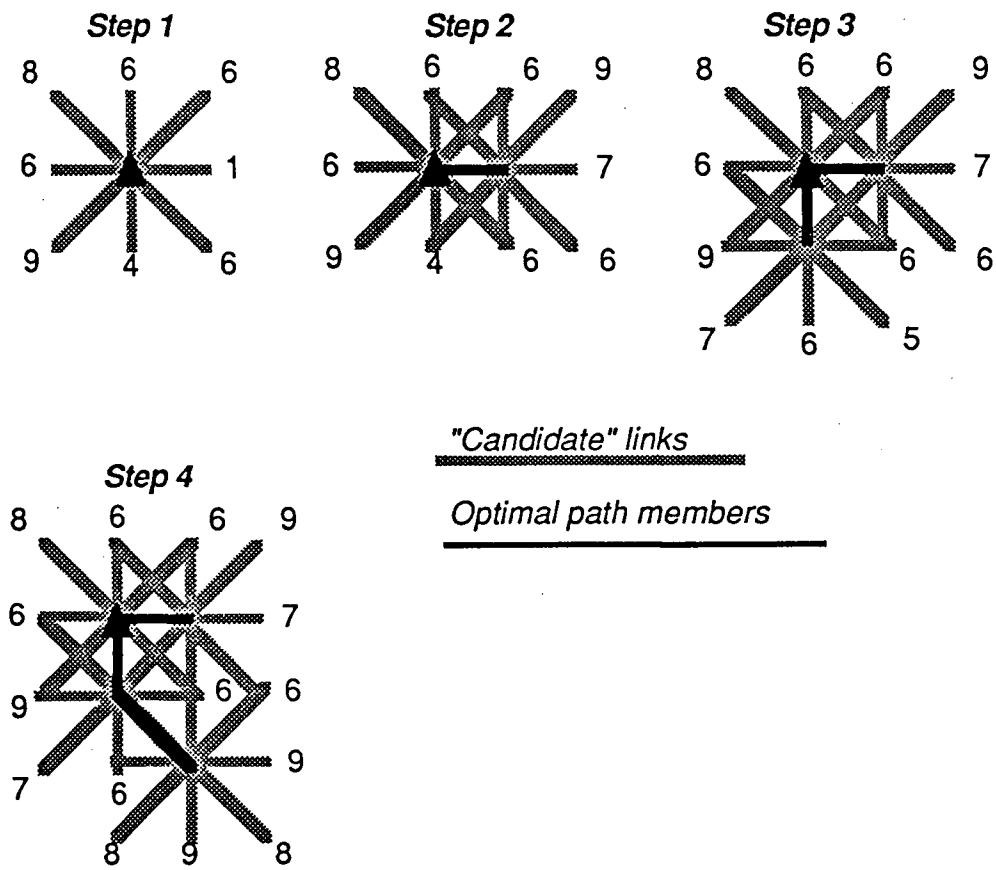


Figure 11. Graph Generated During a Route Search

```
go(P,P).
go(P1,P2) :-
        link(Next,P1,_),
        go(Next,P2).
```

The first clause simply states that to go from a point P to itself, no steps have to be taken. When it is applied, the search is over.

The second clause can be read as "to go from a point P1 to a point P2, find a link that goes from P1 to a third point and then go from that point to P2".

This predicate shows how Prolog is well suited to solve graph related problems. Note that the links are used in reverse order, since the tree is created from the destination to the origin and traversed in the opposite direction.

4.6.2. Enhancements to the Search Routine

Even though the shortest path procedure constitutes an appealing approach, several weaknesses exist. For example, the fact that the study area is "discretized" by superimposing the grid creates geometric distortions that may cause sub-optimal routes to be selected.

The first type of geometric distortion is orientation distortion, whereby the

appearance of the final route depends on how the grid is oriented (Figure 12). Another type is elongation distortion. If only orthogonal moves are allowed, then the shortest possible diagonal route takes two moves and is approximately 41% longer than the actual optimum, a straight line (Figure 12). It should be noted that using a smaller grid will not necessarily reduce this type of distortion. Finally, proximity distortion can cause a path that has highly unsuitable neighbouring cells to be selected over one that has marginally higher impact but is more desirable for being located in a generally low-impact area (Figure 13).

Moreover, it is entirely possible that there exists one or more paths that are quite

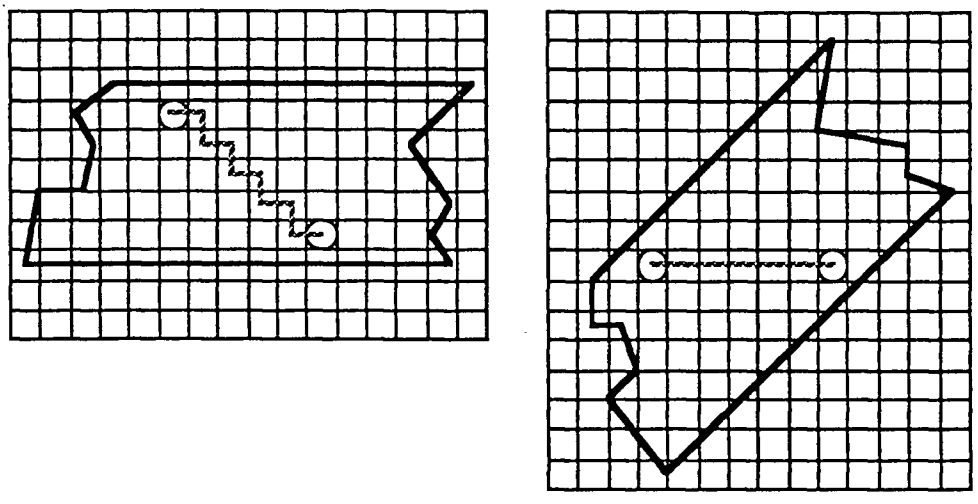


Figure 12. Orientation and Elongation Distortion

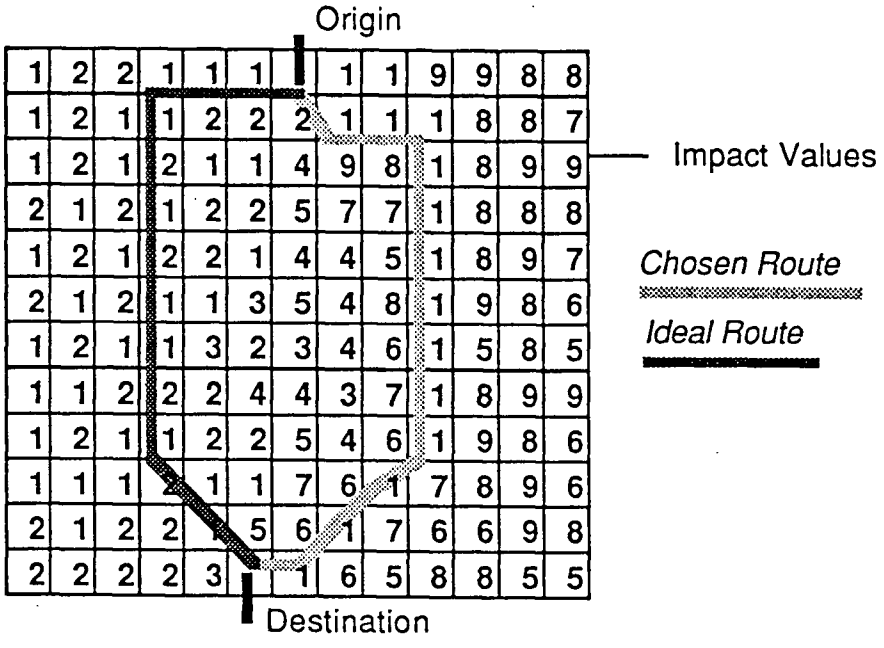


Figure 13. Proximity Distortion

different spatially and yet very close to the optimum in terms of estimated impact values. One remedy to this problem lies in explicitly searching for near-optimal alternate corridors. HLA's search procedure incorporates some features that address these problems.

One of the search parameters is the step size, or grid spacing. By selecting several routes with varying grids, it is possible to identify geometric distortions. Also, the predicate that creates new candidates is extremely flexible, allowing for non-orthogonal links that yield less distorted routes (unfortunately, this type of search involves a significant increase in computational effort).

Since the search procedure is modular, it is easy to utilize different objective functions. This not only enables the user to locate routes according to different criteria but can also indicate the stability of a particular solution. Since HLA measures fuzzy values, obvious candidates for minimization are the highest and lowest possible impacts, expected impact and the uncertainty associated with the answer.

Finally, the routine that generates the links checks whether they lie within a specified search area. By specifying different pairs of search area/grid spacing parameters, it is possible to repeatedly refine the solutions with relative efficiency. This capability also allows the user to focus the analysis on troublesome areas, to specifically exclude regions that are known beforehand to be undesirable, or force the generation of alternate routes for comparison purposes.

The Prolog source code for the predicates related to optimal path search is included in Appendix 1, Partial Program Listing.

4.7. Processing Uncertain Information

Fuzzy numbers are used in HLA to represent uncertain information. The approach differs completely from the probabilistic approach. Strictly speaking, fuzzy models do not take into account the number of ways in which an event can occur, only the

fact that the occurrence may happen.

4.7.1. Definitions

A fuzzy number ***n*** is defined in terms of its membership function $\mu(x)$, which expresses the truth value of the assertion "the value of n is x". A typical membership function is depicted in Figure 14. Membership functions have the following characteristics (using the notation from Figure 14):

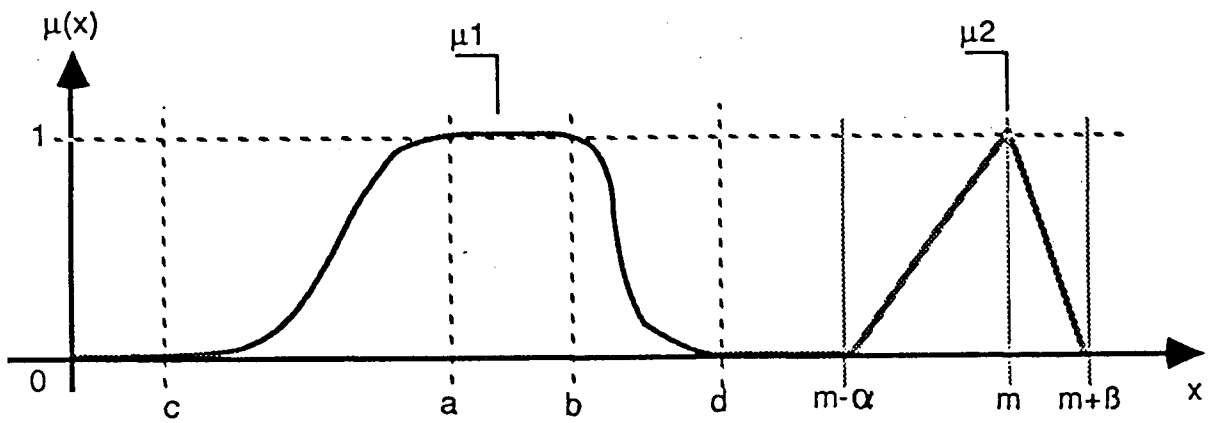


Figure 14. Typical Membership Functions

- A continuous mapping from $\boldsymbol{R}$ to the interval [0,1].
- $\mu(x)=0$ for any value of x from minus infinity to c.
- Strictly increasing on [c,a].
- $\mu(x)=1$ for any value of x in [a,b].
- Strictly decreasing on [b,d].
- $\mu(x)=0$ for any value of x from d to plus infinity.

Two fuzzy numbers are equal if and only if they have the same membership function. Furthermore, a measure of the uncertainty associated with a fuzzy number $\boldsymbol{n}$ is given by the area under the membership function. If the area is zero, then $\boldsymbol{n}$ is a real number. Generalizing, and still using the notation from Figure 14:

- If a=b=c=d, then $\boldsymbol{n}$ becomes an ordinary real number.
- If a=c and b=d, then $\boldsymbol{n}$ represents a tolerance interval [a,b] of the measurement of a quantity.
- If a=b, then $\boldsymbol{n}$ is a representation of a fuzzy number, the value of which is "approximately a".

4.7.2. Fuzzy Number Operators

To use the concept, it is necessary to extend algebraic operations to fuzzy numbers. This can be achieved through application of Zadeh's extension principle (1973). Although the extension principle is well defined, actual implementation and processing of extended operations are not trivial except for relatively simple cases.

The computational aspect of fuzzy information processing has been addressed in the literature [2,11,27,40]. Dubois and Prade [11] introduced efficient and exact operators for the simple algebraic combination of fuzzy numbers with infinite supports.

The four basic operators for strictly positive fuzzy numbers are (as implemented in HLA):

- (m,α,β) [+] $(n,\gamma,\delta) = (m+n,\alpha+\gamma,\beta+\delta)$
- (m,α,β) [−] $(n,\gamma,\delta) = (m-n,\alpha+\gamma,\beta+\delta)$
- (m,α,β) [x] $(n,\gamma,\delta) \simeq (m.n,m.\gamma+n.\alpha,m.\delta+n.\alpha)$
- (m,α,β) [/] $(n,\gamma,\delta) \simeq (m/n,(m.\delta+n.\alpha)/n^2,(m.\delta+n.\beta)/n^2)$

Where fuzzy numbers are represented by three-tuples (m, α,β) as depicted in Figure 14 and the square brackets represent fuzzy operations.

Many other operations can be immediately derived from these. The power function of a fuzzy number, for example, is given by:

(m,α,β) [Power] $p \simeq (m^p,p.m.\alpha,p.m.\beta)$.

It should be noted that uncertainty levels increase very fast when many operations have to be performed on numbers with high associated uncertainty, especially when the [x] or [/] operators are used. For this reason, redundant use of these operators should be avoided, since it can cause losses of precision that would not occur if

the numbers involved were real. In practice, this means that multiplying and dividing a number by a non-zero constant will leave the result unchanged only if the constant is a real number. If the constant involved is a fuzzy number, the result will have higher associated uncertainty. The following example illustrates this:

If X and Y are real numbers and $Y \neq 0$,
then $X \cdot Y / Y = X$

However, if X and Y are fuzzy numbers
$X = (10,0.5,0.2)$ and $Y = (3,0.2,0.4)$,
then $X \cdot Y / Y \simeq (10,2.5,3.17) \neq X$

4.7.3. Implementation Details

The fuzzy operators listed above were implemented in HLA. Frequently used combinations of those were also defined to enhance efficiency and code clarity. These include fuzzy present worth of a series of payments and fuzzy number scaling.

HLA accepts three type of numeric input: ordinary real numbers, tolerance intervals and fuzzy numbers of the (m,α,β) type. Internally, all numbers are converted to the general three-element list format. This is accomplished by the ***fuzzify*** predicate, composed of three clauses. Figure 15 shows how the predicate works.

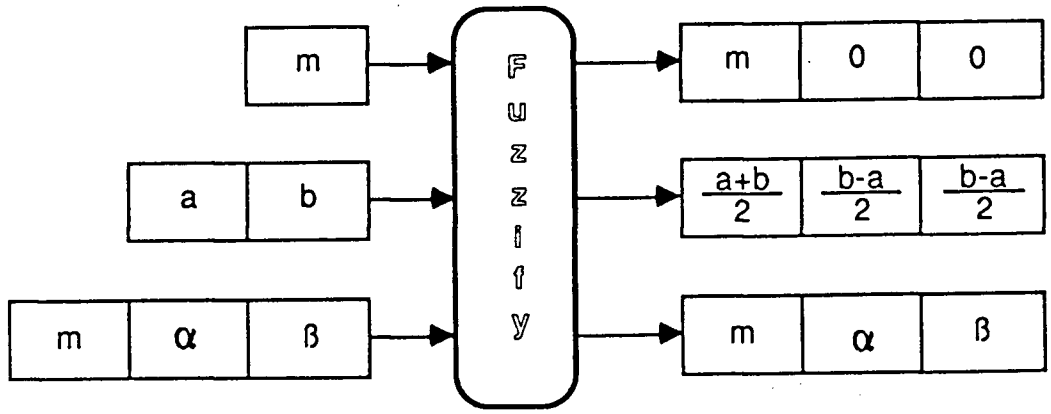


Figure 15. The FUZZIFY predicate

To significantly improve execution speed, all the predicates that operate on fuzzy numbers were compiled. This has the side effect of making their clauses invisible to the rest of the system, and, consequently, unavailable to the explanation predicates. This is not a serious drawback, since explicit information (such as a summary of the article where the operators are derived) is much more helpful in this particular case than a display of the rules employed.

The Prolog source code for all the predicates related to fuzzy number operators is listed in Appendix 1, Partial Program Listing.

5. SAMPLE RUN

The aim of this chapter is to provide an example of how HLA can aid in the solution of highway corridor location problems. All the capabilities of the system cannot be demonstrated in one example, but the most significant features are demonstrated in this chapter. The interaction between HLA's modules is also illustrated.

The present example is based on an actual route location/evaluation report, the contents of which are proprietary. It involves the evaluation of three alternate routes between two towns located in a study area of approximately 1600 square kilometres. In the example, the proposed routes were analyzed and compared using HLA and a subset of the data in the original report. An optimal route was also generated and compared with the proposed alternatives.

The original report involved six different teams of consultants. The evaluation approach adopted involved the use of subjective ratings in a three-point scale (good, moderate, poor) for a variety of impact dimensions (24, to be exact). The final conclusions were subjectively drawn from these disaggregated ratings.

Figure 19 shows the original study area and the three proposed routes. The towns are indicated as points ***A*** and ***C***, and an intermediate point ***B***, through which all routes must pass, is also marked.

5.1. Tabulating the Data

The first step in the analysis consists of coding the data in HLA format. Maps of the study area are utilized to build a digital model of the region. Relatively homogeneous subareas are defined and grouped into sub-maps. Any number of sub-maps may be generated, each one conveying different types of information. In the present example three sub-maps were defined to represent regional geology, land use and population density, and topography and existing highways. These are depicted in Figures 16, 17 and 18, respectively. The corresponding Prolog clauses which define these areas are listed in Appendix 1. The population densities were specified as fuzzy values, either to account for expected fluctuations or to reflect uncertainty in the data.

It should be noted that, although conceptually simple, coding of the regional data is a very error-prone process. A digitizing tablet would be ideal for entering this type of information and some type of graphical output is indispensable for checking the consistency of the data.

5.2. Evaluating the Proposed Routes

Evaluation of proposed routes is accomplished in a straightforward manner. Once all

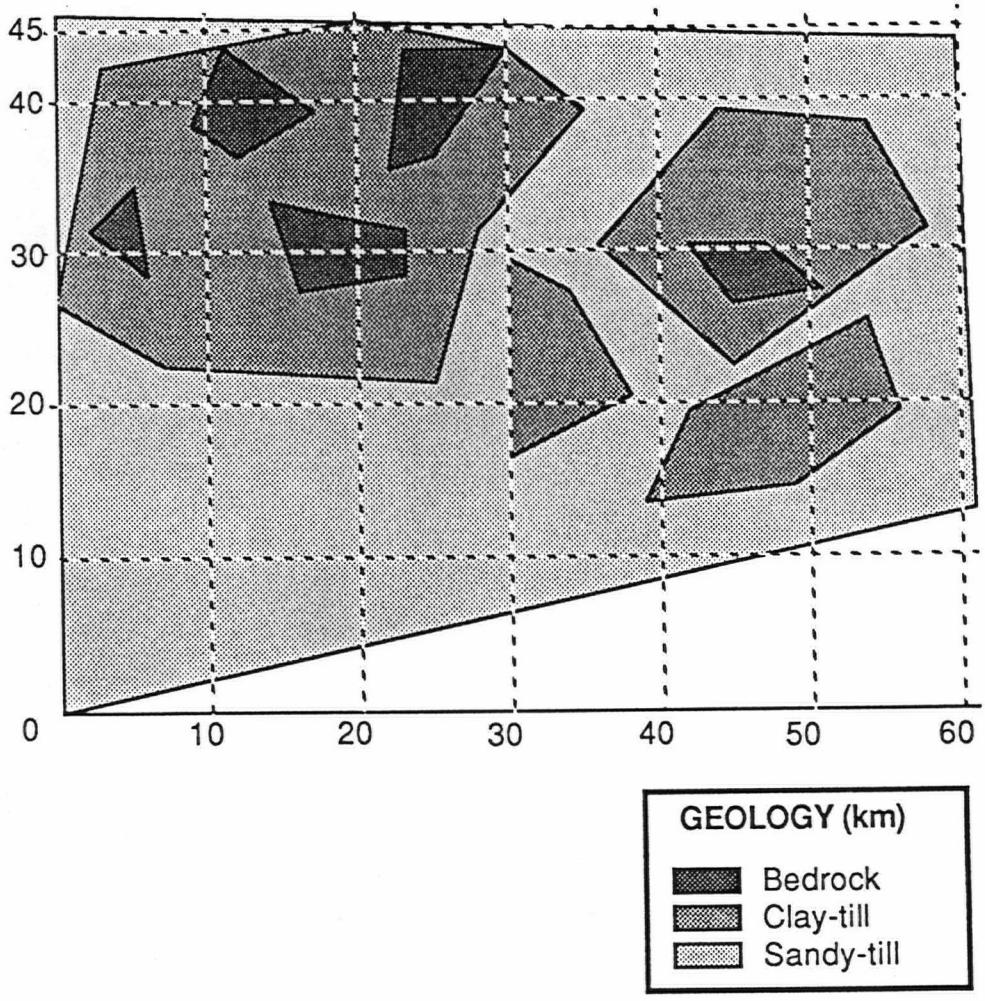


Figure 16. Geology Model

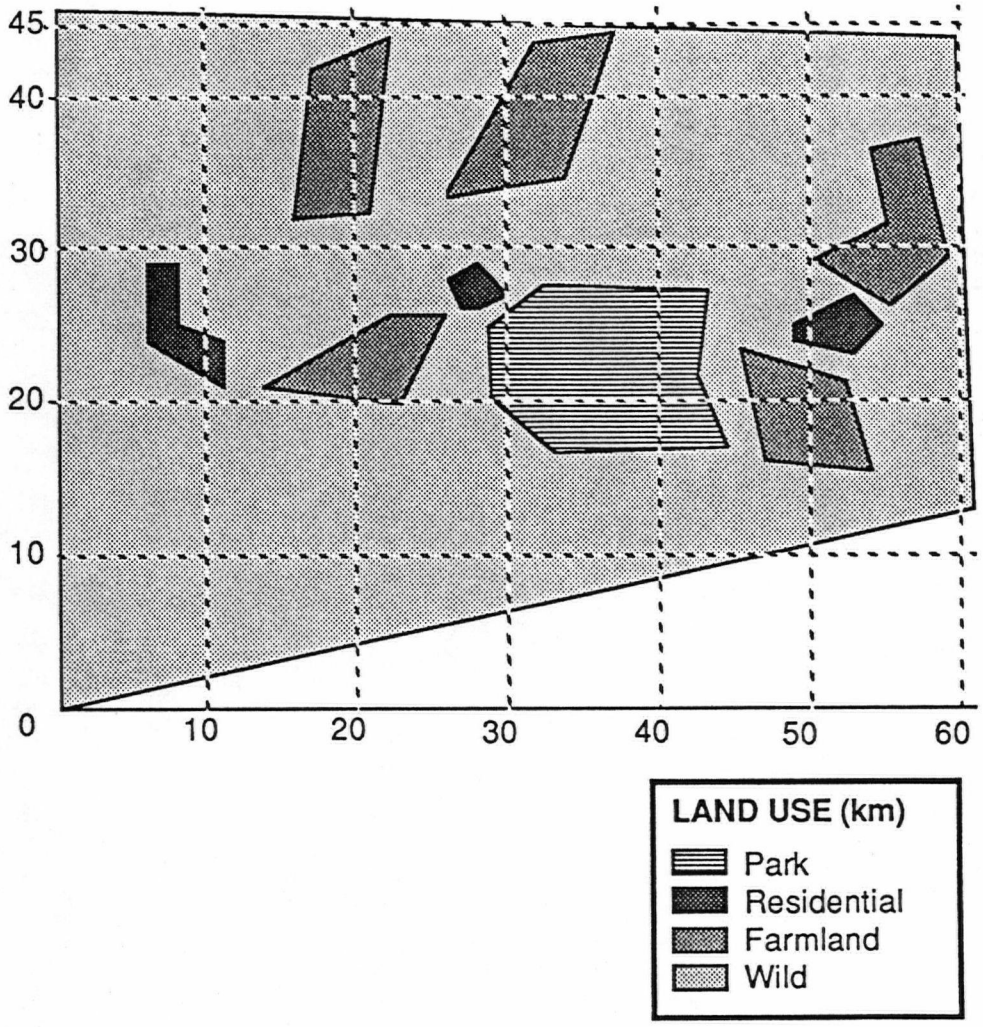


Figure 17. Land Use and Population Density Model

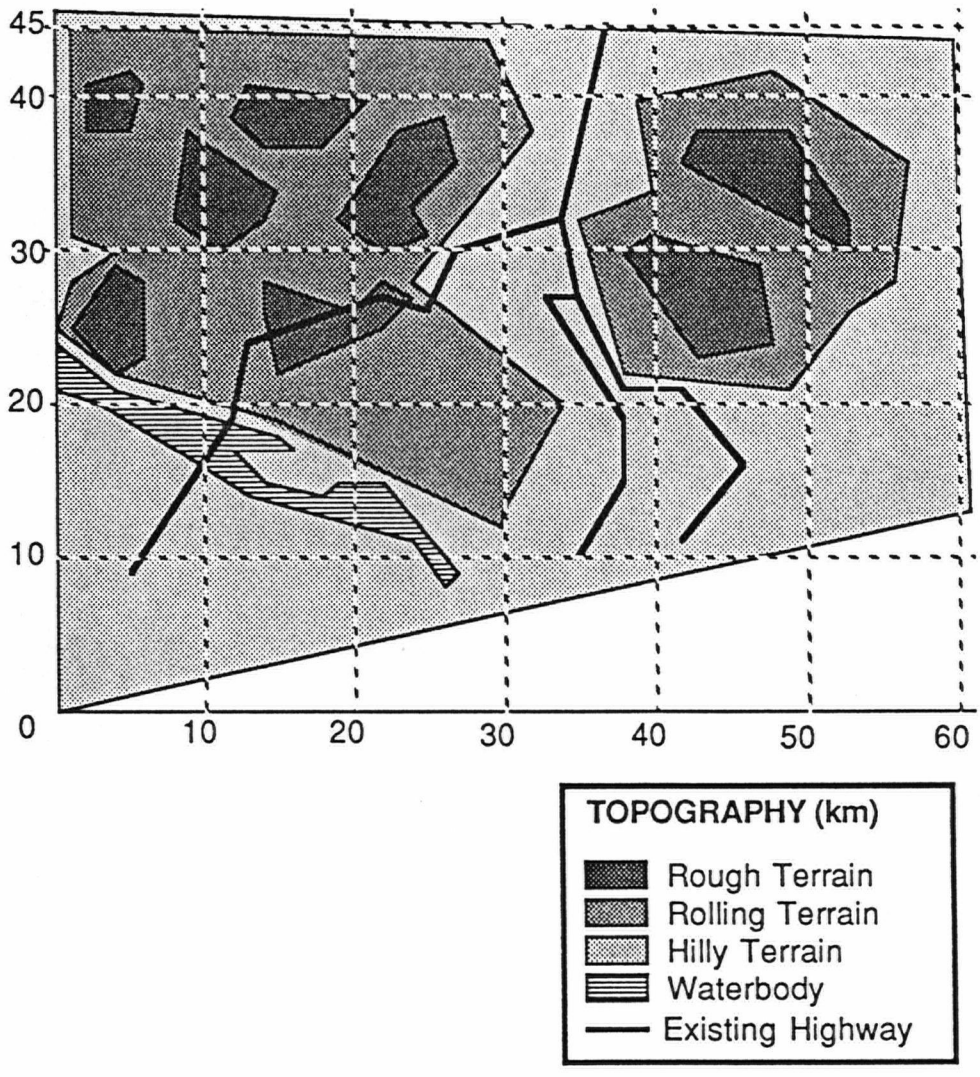


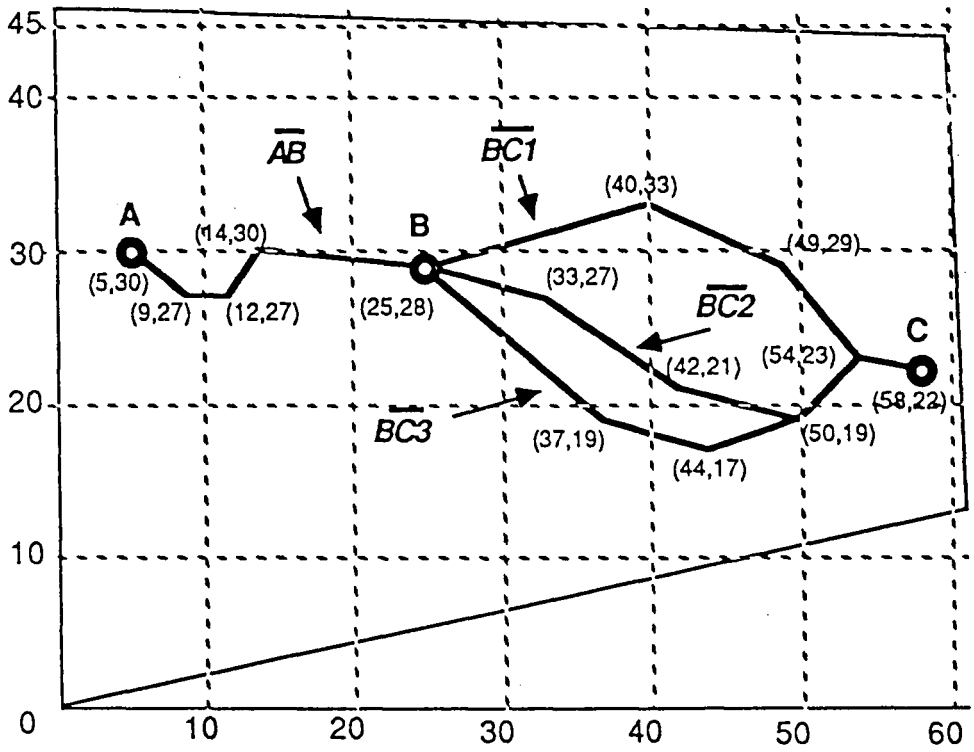
Figure 18. Topography and Existing Highways Model

the project information has been entered, all that is required is one call to the predicate ***solve(analyze_route(Step_Size,Route))*** The variable ***Route*** should contain an open polygon (defined as a list of points) representing the proposed alignment. Figure 19 shows the four routes (AB, BC1, BC2 and BC3) evaluated in this example. During the analysis, each line segment is broken into smaller ones with lengths of approximately ***Step_Size***, and these segments are evaluated one by one.

The output provided consists of a list of disaggregated impacts for each line segment and a global aggregated impact value for the whole route. The lists of disaggregated impacts can be used to build impact profiles for each route. Figures 20 and 21 represent the impact profiles for routes AB and BC1. This type of diagram clearly shows the relative significance of each type of impact and critical points (in terms of impact) along the route.

The global impact values obtained for each alignment can be found in Figure 19. It is possible to conclude, based on these numbers, that the best route between points A and C is AB-BC3. Since the values provided are fuzzy, it is also possible to assess the uncertainty associated with the answers. Among the routes analyzed, the one with lowest associated uncertainty is AB-BC2. It is important to note that these results coincide with the ones in the original report, which also concluded AB-BC3 was the best suggested alignment.

A call of the form ***analyze_route(Step_Size,Route)*** would yield the same results as ***solve(analyze_route(Step_Size,Route))*** with increased efficiency, but HLA's inference



Suggested Routes			
Route	*Estimated Impact*		
	m	α	ß
AB	57.07	7.11	7.89
BC1	96.35	11.84	13.39
BC2	83.37	11.69	12.90
BC3	82.66	11.88	12.98

Figure 19. Proposed Routes

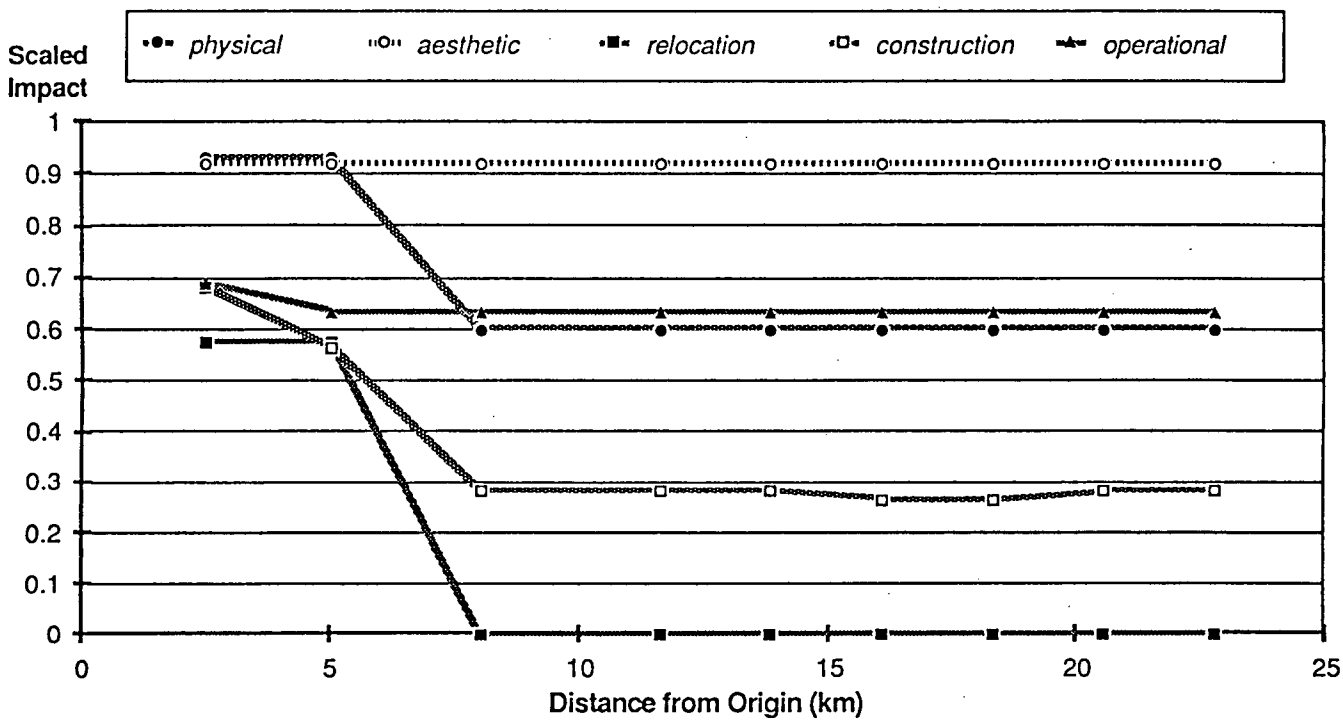


Figure 20. Impact Profiles for Proposed Route AB

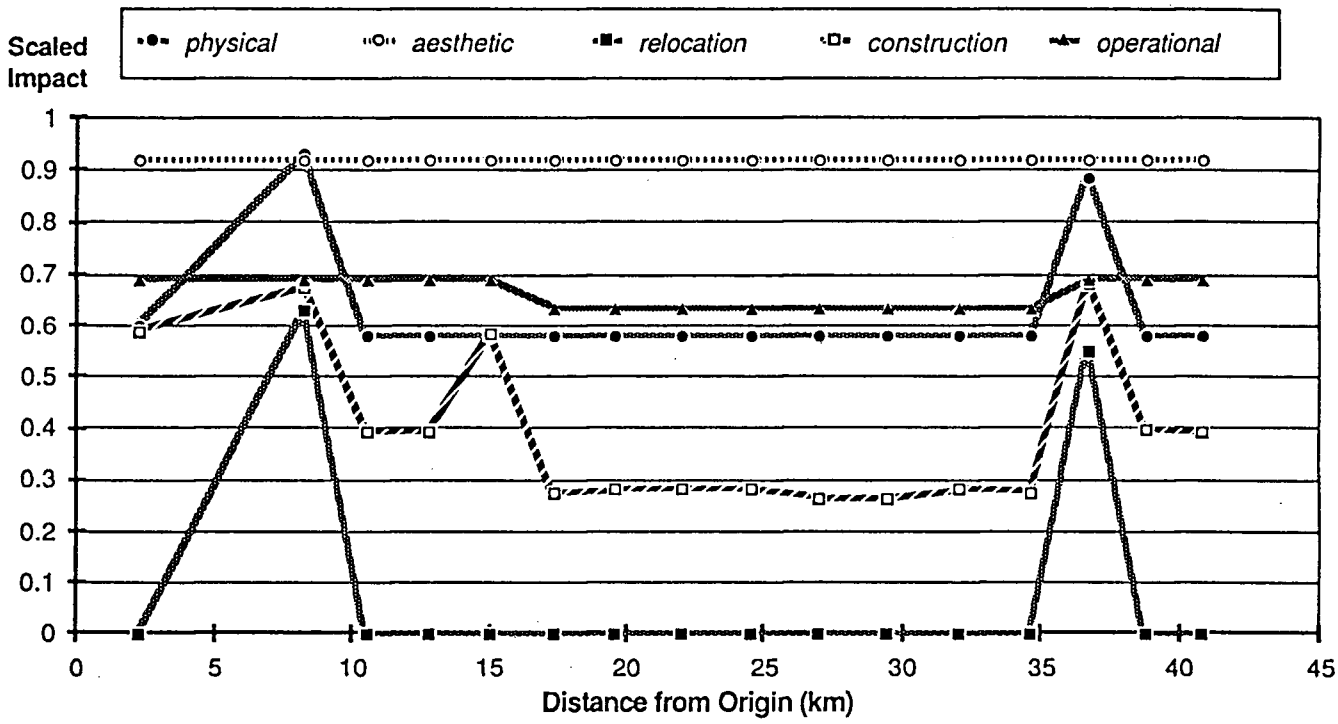


Figure 21. Impact Profiles for Proposed Route BC1

engine would be bypassed. The advantage of using the ***solve*** predicate is that it can prompt the user for eventual missing values and provide "why?" explanations. A typical interaction between HLA and the user is depicted in Appendix 2. For clarity, the user responses are reproduced in bold-faced characters.

The explanation facilities can be used to examine how impact coefficients were assessed. This capability makes the underlying assumptions and methods employed explicit, enabling experienced users to check their applicability and correctness. It can also be used to document the route selection process or as a training tool for novices. Figure 20 shows that point p(28.21,29.07) has high associated impact in comparison with neighbouring points. The ***how*** predicate can be used to investigate this situation, as shown in Appendix 3.

5.3. Searching for an Optimal Route

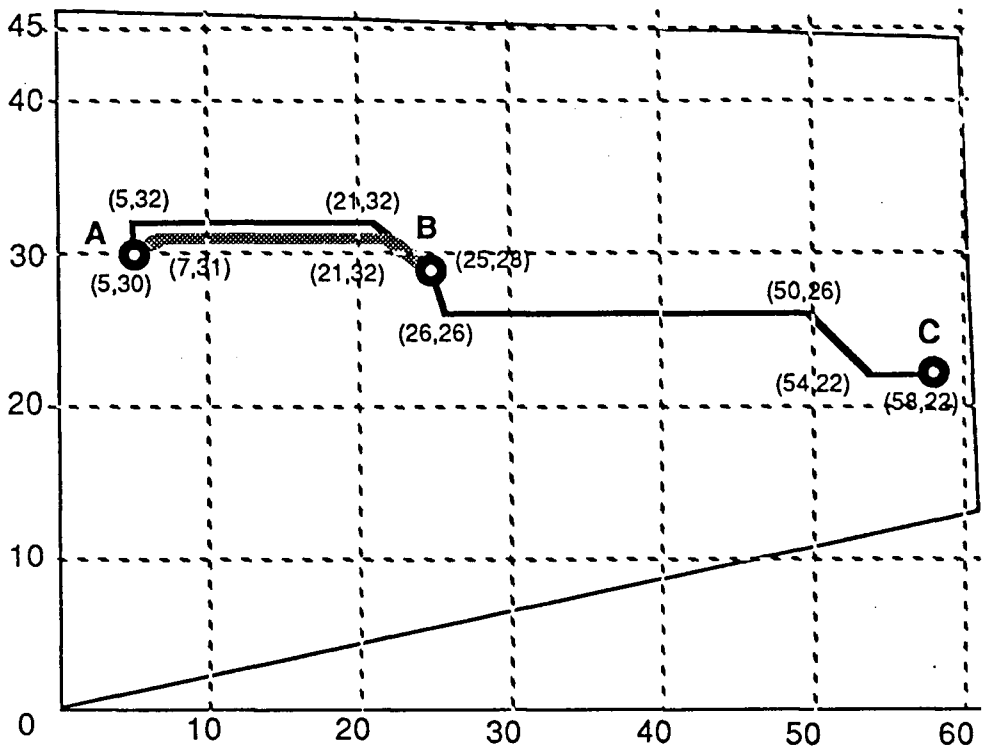
The Search module employs the evaluation predicates described above and a shortest path algorithm to locate optimal alignments. To determine a route with minimum aggregated impact, a call of the form ***search_path(Step_Size,Origin,Dest)*** is used. Two optimal routes were determined between points A and B, with step sizes of 3 and 4 kilometres. The route obtained with shorter step size is closer to the real optimum. The 3 km step size route has an associated impact almost 16% lower than the one obtained with a 4 km step size or the proposed alignment. An

optimal route was also obtained between points B and C, with a 4 km step size. Even with this relatively large step size, the impact for the route located by HLA is more than 7% lower than the one associated with the best proposed route between points B and C (BC3). The optimal routes with corresponding step sizes and impact estimates are depicted in Figure 22.

5.4. Conclusions

The results obtained using HLA should not be directly compared to the ones in the original report, since distinct evaluation approaches were employed and the data used by HLA represents only a subset of the information in the report. Nevertheless, it was extremely gratifying to notice how the results obtained using HLA were acceptable and comparable to the ones found in an actual route evaluation report, especially since the system is currently in a prototype stage.

All the information available was coded into an HLA format Project Description File (PDF) with remarkable ease, including physical maps and uncertain data. The flexible, interactive environment provided by HLA then enabled large amounts of useful information to be extracted from the raw data file quickly and easily. This not only supplied answers to specific questions (such as "how do these routes compare" or "what is the minimal impact alignment") but also, in a relatively short time, provided the analyst with a good insight of the problem.



Minimal Impact Routes

Route	*Step Size (km)*	*Estimated Impact*
A-B	4	56.96
A-B	3	47.94
B-C	4	76.55

Figure 22. Optimal Routes for Various Step Sizes

6. FURTHER RESEARCH REQUIREMENTS

Due to the time constraints associated with this project, a balance had to be established between the number of features provided by HLA and the final size of its knowledge base. Even though the refinement of existing rules and addition of new ones could turn HLA into a full-fledged production system (and this is the most obvious expansion path for additional research), the framework it provides is far from exhaustive. Totally new capabilities could be incorporated into the system to enhance its usefulness.

It would be desirable to add rules to suggest and evaluate the effectiveness of mitigation measures along proposed routes. Mitigation measures are employed to minimize certain types of impacts highways can cause. Examples include noise barriers and the construction of artificial lakes to hide visually unpleasant borrow pits.

The fuzzy logic approach could be logically extended to the spatial data representation structures, allowing for areas with fuzzy borders to be specified. This feature would reflect the fact that, in reality, some sub-region boundaries are not well defined. The extent to which this capability would affect the answers provided by the system, however, is not evident at the moment.

Several optimization techniques could be used to increase HLA's execution speed

without reducing its flexibility. The simplest one would be to use more efficient implementation tools, software and hardware, as they become available. To illustrate the potential of this approach, it should be mentioned that certain modules (particularly the geometry related predicates) of HLA were developed in a non-standard implementation of Prolog that sacrifices some of the language's features to achieve execution speeds in the order of twenty times faster than standard interpreters. Another option would be to implement some predicates, particularly the procedural ones, in more efficient, conventional languages.

Because Prolog provides extensive flexibility and modularity, documentation assumes an even more important role than in traditional environments. Self-documenting capabilities would be of great assistance during the development of systems that take advantage of these characteristics. For example, a relatively simple system to list the knowledge base graphically, in the form of inference networks (such as the one depicted in Figure 4), would be extremely useful for documentation purposes. Such a system could also be used by the explanation facilities to provide the user with information in a more comprehensive fashion.

The bottleneck in the development of expert systems has been the acquisition and representation of the knowledge. Efforts directed towards self-learning and automatic knowledge organization capabilities of systems can provide a proficient method of dealing with the large knowledge bases required. General mechanisms that enable systems to learn from their own experiences and to organize the knowledge they embody would represent a quantum leap not only in the development of

knowledge-based systems but in the whole field of Artificial Intelligence.

7. CONCLUSIONS

Knowledge-based systems can help solve problems that conventional computer programs could never solve, and also provide a new way of looking at traditional problem-solving methods. These systems are not necessarily better than the traditional ones, in practice they tend to be complementary. This fact is reflected by the current popularity of the knowledge-based approach and seems to guarantee that application of such systems will increase significantly in the near future.

The system described in this paper shows that knowledge based systems do not have to be limited to the solution of comparatively simple selection problems. They can be effectively applied to most kinds of ill-structured problem. To fully realize this potential, the flexibility of a full-featured high level language is preferable over the easier to use expert system shells. These can be convenient in a few particular cases (e.g. for prototyping the knowledge-base), but will usually impose serious limitations upon the system designer. HLA, for example, relies heavily on Prolog's features to create and manipulate specialized, dynamic data structures. It also combines knowledge-based modules (evaluation predicates) with conventional ones (e.g. search procedure). Currently available shells would not support either of these characteristics.

It is worthwhile to note that the type of knowledge HLA embodies represents limited expertise in a variety of fields. This apparently contradicts the paradigm that

such systems should contain extensive knowledge about relatively restricted domains. However, HLA is not a typical knowledge-based system. It can handle usual problems and identify potential difficulties that would require the presence of genuine experts. Furthermore, highway location practitioners are not experts in all of the aspects involved. Their primary task is to determine what information is relevant and to combine it in a useful and comprehensive way.

The language used to develop HLA, Prolog, proved to be a good development tool. All the facilities usually expected from expert systems were easily implemented without sacrificing flexibility. The ease with which procedural tasks were accomplished in an essentially declarative language was also very important in this prototyping phase.

A few warnings are in order. Prolog—and most other "fifth-generation" languages—can be rather inefficient in terms of speed when compared to traditional languages. While it can be used to implement entire prototypes, production systems should restrict its use to the tasks it can accomplish best. HLA could be optimized for speed by an order of magnitude by rewriting critical procedural routines in a traditional language (most Prolog implementations offer ways of linking routines written in C, Pascal or assembly language). Also, the almost unlimited flexibility provided places great responsibility upon the system designer. Since the language allows an almost unlimited variety of design solutions, the task of keeping the software maintainable and consistent is assigned almost solely to the designer. Conventional implementation tools, on the other hand, tend to impose rigid

guidelines that, at the cost of reducing flexibility, alleviate this burden substantially.

Knowledge-based systems can be extremely powerful and flexible tools, applicable to a variety of problems. However, they are essentially models of the human expertise available and therefore cannot provide magical solutions. The level of performance of such systems tends to be proportional to the amount of time and effort spent in developing their knowledge base. Currently, production systems take around five to ten man-years to develop.

BIBLIOGRAPHY

A. Artificial Intelligence and Expert Systems

1. Aikins, J. S. (1983). "Prototypical Knowledge for Expert Systems." ***Artificial Intelligence***, Vol. 20, No. 2, pp. 163-210.

2. Bandler, W. and Kohout, L. J. (1985). "Probabilistic Versus Fuzzy Production Rules in Expert Systems." ***International Journal of Man-Machine Studies***, No. 22, pp. 347-353.

3. Barr, A. and Fiegenbaum, E. A. (1982). ***The Handbook of Artificial Intelligence***, Vol. 1, pp. 3-43, 143-216, Kaufman Inc.

4. Benoit, J. W.; Davidson, J. R. and Powell, E. G. (April 1986). "Lessons Learned Developing an Expert System." ***Application of Artificial Intelligence in Engineering Problems***, First International Conference, Southampton University, U. K., pp. 77-83.

5. Bonissone, P. P. and Tong, R. (1985). "Editorial: Reasoning with Uncertainty in Expert Systems." ***International Journal of Man-Machine Studies***, No. 22, pp. 241-250.

6. Charniak, E. and McDermott, D. (1985). ***Introduction to Artificial Intelligence***. Addison-Wesley.

7. Clancey, W. J. (1983). "The epistemology of a Rule-Based Expert System—a Framework for Explanation." ***Artificial Intelligence***, Vol. 20, No. 3, pp. 215-251.

8. Clark K. L. and McCabe, F. G.(1982). "PROLOG: a Language for Implementing Expert Systems." ***Machine Intelligence***, No. 10, pp. 455-470.

9. Clocksin, W. F. and Mellish, C. S. (1981/84). ***Programming in PROLOG***, Springer-Verlag.

10. Dankel, D. D. (February 1986). "Expert Systems: Misconceptions and Reality." ***Artificial Intelligence***, Society of Automotive Engineers, International Congress and Exposition, Detroit, Michigan, pp. 1-4.

11. Dubois, D. and Prade, H. (1978). "Operations on Fuzzy Numbers." ***International Journal of Systems Science***, Vol. 9, No. 6, pp. 613-626.

12. Duda, R. O. and Gashing, J. G. (1985). "Expert Systems Come of Age." ***Applications in Artificial Intelligence***, Petrocelli books, Princeton, NJ.

13. Duda, R. O. and Shortliffe, E. H. (1983). "Expert Systems Research." ***Science***, Vol. 220, April. 1983, pp. 261-268.

14. Fenves, S. J. et al (1984). "Knowledge-based Expert Systems in Civil Engineering." ***Computing in Civil Engineering***, ed. Hodge, pp. 249-257.

15. Garvey, T. D.; Lowrence, J. D. and Fischler, M. A. (1981). "An Inference Technique for Integrating Knowledge from Disparate Sources." ***Proc. IJCAI***, No. 7, pp. 319-325.

16. Gevarter,W. (1985)."Expert Systems: Limited but Powerful." ***Applications in Artificial Intelligence***, Petrocelli books, Princeton, NJ.

17. Hayes, R. F.; Waterman, D. and Lenat, D. (1984). ***Building Expert Systems***, Addison-Wesley Co., pp. 129-167.

18. Ishizu, K. A. M.; Fu, S. and Yao, J. T. P. (1982). "Inference Procedures Under Uncertainty for the Problem Reduction Method." ***Information Science***, No. 28, pp. 179-206.

19. Joyce, J. D. and Uthurusamy, R. (February 1986). "Promises and Pitfalls of Knowledge Systems: an Overview." ***Artificial Intelligence***, Society of Automotive Engineers, International Congress and Exposition, Detroit, Michigan, pp. 1-4.

20. Karp, R. M. and Pearl, J. (1983). "Searching for an Optimal Path in a Tree with Random Costs." ***Artificial Intelligence***, Vol. 21, Nos. 1-2, pp. 99-116.

21. Lehner, P. E. and Barth, S. W. (1985). "Expert Systems on Microcomputers." ***Applications in Artificial Intelligence***, Petrocelli books, Princeton, NJ.

22. Lesmo, L.; Saita, L. and Torasso, P. (1985) "Evidence Combination in Expert Systems." ***International Journal of Man-Machine Studies***, No. 22, pp. 307-326.

23. Lloyd, J. W. (1984). ***Foundations of Logic Programming***, Springer-Verlag.

24. Martin-Clouaire, R. and Prade, H. (1985). "On the Problems of Representation and Propagation of Uncertainty in Expert Systems." ***International Journal of Man-Machine Studies***, No. 22, pp. 251-264.

25. Maher, M. L. (1986). "Problem Solving Using Expert System Techniques." ***Expert Systems in Civil Engineering***, ASCE, New York, pp. 7-17.

26. Merritt, D. (1986). "Forward Chaining in Prolog." ***AI Expert***, November issue, pp. 30-42.

27. Ogawa, H.; Fu, K. S. and Yao, J. T. P. (1985). "An Inexact Inference for Damage Assessment of Existing Structures." ***International Journal of Man-Machine Studies***, No. 22, pp. 295-306.

28. Pearse, R. and Rosenbaum, M. (April 1986). "The Evaluation of Road Corridors by the Use of an Expert System." ***Application of Artificial Intelligence in Engineering Problems***, First International Conference, Southampton University, U. K., pp. 719-730.

29. Quinlan, J. R. (1983). "INFERNO: a Cautious Approach to Uncertain Inference." ***The Computer Journal***, No. 26, Vol. 3, pp. 255-269.

30. Shafer, G. (1976). ***A Mathematical Theory of Evidence***, Princeton, NJ: Princeton University Press.

31. Shapiro, E. Y. (1983). "Logic Programs with Uncertainties: A Tool for Implementing Rule-Based Expert Systems." ***Proceedings of the IJCAI***, No. 8, pp. 529-532.

32. Stefik, M. et al. (1982). "The Organization of Expert Systems." ***Artificial Intelligence***, Vol. 18, No. 2, pp. 135-173.

33. Sterling, Leon and Shapiro, Ehud. (1986). ***The Art of Prolog***, The MIT Press, pp. 303-330.

34. Sterling, Leon. (1984). "Expert System = Knowledge + Meta-Interpreter." ***The Weizmann Institute of Science, Department of Applied Mathematics***, Rehovot, Israel.

35. Swartout, W. R. (1983). "XPLAIN: a System for Creating and Explaining Expert Consulting Programs." ***Artificial Intelligence***, Vol. 21, Nos. 1-2, pp. 285-325.

36. Walker, A. (1983). "Prolog/EX1, An Inference Engine Which Explains Both Yes and No Answers." ***Proceedings of the IJCAI***, No. 8, pp. 526-528.

37. Waterman, D. A. (1986). ***A Guide to Expert Systems***. Addison-Wesley.

38. Wigan, M. R. (July 1986). "Engineering Tools for Building Knowledge-Based Systems on Microsystems." ***Microcomputers in Civil Engineering***, Vol. 1, No. 1.

39. Whalen, T. and Schott, B. (1985). "Alternative Logics for Approximate Reasoning in Expert Systems: a Comparative Study." ***International Journal of Man-Machine Studies***, No. 22, pp. 327-346.

40. Wong, F. S. and Dong, W. (April 1986). "Fuzzy Information Processing in Engineering Analysis." ***Application of Artificial Intelligence in Engineering Problems***, First International Conference, Southampton University, U. K., pp. 247-255.

B. Highway Location and Evaluation

41. Arditi, D. (1984). "Railway Route Rationalization: A Validation Model." ***Transportation Research Record***, No. 948, pp. 105-113.

42. Civco, D. L.; Kennard, W. C. and Lefor, M. Wm. (1980). "Computer Assisted Highway Corridor Location." ***Technical Paper of the American Society of Photogrammetry***, Congress of Survey and Mapping, pp. 323-332.

43. Cohn, L. F. and McVoy, G. R. (1982). ***Environmental Analysis of Transportation Systems***. New York: John Wiley & Sons.

44. Dijkstra, E. W. (1959). "A Note on Two Problems in Connection with Graphs." ***Numerical Mathematics***, No. 1, pp. 269-271.

45. Dooley, J. E.; Newkirk, R. T. (1976). "A Planning System to Minimize Environmental Impact Applied to Route Selection." ***Kybernetes***, Vol. 5, pp. 213-220.

46. Elsner, G. H.; Travis, M. R. and Kourtz, P. Z. (1975). "Dynamic Programming Subroutines Based on the Dijkstra Algorithm for Finding Minimum Cost Paths in Directed Networks." ***Information Report FF-X-51***, Forest Fire Res. Inst., Canadian For. Serv., Dept. Environment, Ottawa, Ontario, 16 pp.

47. Environment Canada. (1978). ***Report of the environmental assessment panel—Shakwak project***. Federal Environmental Assessment Review Office, Vancouver, BC.

48. Freeman, J. (1974). "Regional Transportation Corridor Economic Evaluation Model." ***The Transport Group***, University of Waterloo, Waterloo, Ontario, 119 pp.

49. Gamble, H. B. and Avinroy, T. B. (1978). "Beneficial Effects Associated with Freeway Construction: Environmental, Social and Economic." ***Transportation Research Board***, special report 193, Washington, D. C.

50. Giles, R. H.; Jones, A. B. III; Smart, C. W. (1976). "POWER: A Computer System for Corridor Location." ***Department of Fisheries and Wildlife Sciences, Virginia Polytechnic Institute and State University***, Research Bulletin 117, Blacksburg, VA, 30 pp.

51. Hensher, D. A. et al. (1983). "The Appraisal Process in Transportation: an Australian Experience." ***The Practice of Transport Investment Appraisal***, Edited by K. J. Button and A. D. Pearman, pp. 108-132.

52. Huber, D. L. and Church, R. L. (1985). "Transmission Corridor Location Modelling." ***Journal of Transportation Engineering***, Vol. 111, No. 1, pp. 114-130.

53. Keeney, R. L. and Raiffa, H. (1976). ***Decisions with Multiple Objectives: Preference and Value Tradeoffs***, New York: John Wiley and Sons.

54. Leopold, L. B. et al. (1971). "A Procedure for Evaluating Environmental Impact." ***U. S. Geological Survey Circular***, No. 645, 13 pp.

55. McHarg, I.(1968). "A Comprehensive Highway Route Selection Method." ***Highway Research Record***, No. 246, pp. 1-15.

56. Manheim, M. L. (1979). ***Fundamentals of Transportation Systems Analysis—Vol. 1: Basic Concepts***. Cambridge, Massachussets: The MIT Press.

57. Massam, Bryan. (1979). "Highway Routing: An Alternative Method for Comparing Alternatives." ***Joint Program in Transportation, University of Toronto, York University***, report No. 56.

58. Massam, Bryan. (July 1978) "The Search for the Best Alternative Using Multiple Criteria; Singapore Transit Study." ***Economic Geography***, Vol. 54, No. 3, pp. 245-253.

59. Miller, G. A. (1956). "The Magical Number Seven, Plus or Minus Two." ***The Psychological Review***, March issue.

60. Navin, F. P. D. (July 1985). A Preliminary Investigation of the Kingsvale to Aspen Grove Highway Corridor for the Nicola Stock Breeders' Association, ***unpublished***.

61. Nijkamp, P. (1975). "A Multi-Criteria Analysis for Project Evaluation." ***Papers of the Regional Science Association***, Vol. 35, pp. 87-11.

62. Nijkamp, P. (1981). "Qualitative Evaluation Models with Conflicting Priorities." ***Papers of the Regional Science Association***, Vol. 46, pp. 3-14.

63. Nijkamp, P. (1979). "The Use of Psychometric Techniques in Evaluation Procedures." ***Papers of the Regional Science Association***, Vol. 42, pp. 119-138.

64. Nijkamp, P. (1979). ***Multidimensional Spatial Data and Decision Analysis***, Wiley, New York.

65. Oglesby, C. H. and Hicks, R. G. (1982) ***Highway Engineering***. John Wiley and Sons: New York, fourth edition.

66. Pearman, A. D. (1983). "Risk and Uncertainty in Road Investment Appraisal." ***The Practice of Transport Investment Appraisal***, Edited by K. J. Button and A. D. Pearman, pp. 156-180.

67. Perrod, P. et al. (1971). "Application of Modern Methods (with Special Reference to Planning, Programming, Budgeting Techniques) to the Choice of Investment Projects." ***Report on the Tenth Round Table on Transport Economics***, European Conference of Ministers of Transport, 79 pp.

68. Roy, B., Sussman, B. and Bouayoun, R. (1966). "A Decision Method in Presence of Multiple View-Points: ELECTRE." ***Paper Presented at the Study Sessions on Methods of Calculation in the Social Sciences***, Rome.

69. Sedgewick, Robert. (1984). ***Algorithms***, Addison-Wesley, pp. 307-319.

70. Sharp, C. H. (1983). "Time in Transport Investment." ***The Practice of Transport Investment Appraisal***, Edited by K. J. Button and A. D. Pearman, pp. 181-195.

71. Shuldiner, P. W.; Cope, D. F. and Newton, R. B. (1979). "Ecological Effects of Highway Fills on Wetlands." ***National Cooperative Highway Research Program Report***, No. 218A, Transportation Research Board.

72. Skinner, R. E. Jr. and Tischer, M. L. (1980). "Environmental and Social Impact Analysis." ***Transportation Planning and Policy Decision Making***, Praeger, New York.

73. Van Delft, A. and Nijkamp, P. (1976). "A Multi-Objective Decision Model for Regional Development, Environmental Quality Control and Industrial Land Use." ***Papers of the Regional Science Association***, Vol. 36, pp. 35-57.

Appendix 1. Sample HLA Data File

```
/* Geology Terrain Model */

area([
        p(11.18,45.11), p(17.16,40.43), p(12.48,37.18),
        p( 9.23,39.91), p(11.18,45.11)],
        [geology(bedrock)]).
area([
        p(23.40,44.72), p(30.68,44.72), p(25.22,37.18),
        p(22.36,36.40), p(23.40,44.72)],
        [geology(bedrock)]).
area([
        p( 5.07,35.75), p( 6.24,29.38), p( 1.95,32.50),
        p( 5.07,35.75)],
        [geology(bedrock)]).
area([
        p(14.04,34.45), p(23.40,32.50), p(23.40,29.25),
        p(16.38,28.47), p(14.04,34.45)],
        [geology(bedrock)]).
area([
        p(42.77,31.33), p(47.97,31.33), p(51.74,28.34),
        p(46.15,27.43), p(42.77,31.33)],
        [geology(bedrock)]).

area([
        p( 2.86,43.68), p(20.41,46.93), p(30.68,44.72),
        p(35.36,40.56), p(28.34,32.50), p(25.22,22.36),
        p( 6.63,23.27), p(-0.26,27.43), p( 2.86,43.68)],
        [geology(clay_till)]).
area([
        p(37.05,31.33), p(45.11,40.56), p(54.99,39.65),
        p(59.41,32.63), p(46.02,23.27), p(37.05,31.33)],
        [geology(clay_till)]).
area([
        p(30.81,30.29), p(34.84,28.08), p(38.61,21.06),
        p(30.81,17.16), p(30.81,30.29)],
        [geology(clay_till)]).
area([
        p(40.04,14.04), p(42.90,20.15), p(55.38,26.13),
        p(57.20,20.15), p(50.18,15.34), p(40.04,14.04)],
        [geology(clay_till)]).

area([
        p( 0.00, 0.00), p( 0.00,47.32), p(61.36,45.24),
```

```
        p(62.27,13.65), p( 0.00, 0.00)],
        [geology(sandy_till)]).

/* Land Use Terrain Model */

area([
        p(28.99,25.74), p(32.89,28.73), p(44.07,28.34),
        p(43.29,22.75), p(45.24,17.81), p(33.80,17.42),
        p(29.38,21.32), p(28.99,25.74)],
        [land_use(park),density([12,17])]).

area([
        p( 5.72,30.29), p( 7.93,30.16), p( 8.06,25.74),
        p(11.05,24.70), p(11.05,21.84), p( 5.98,24.83),
        p( 5.72,30.29)],
        [land_use(urban),density([55,60])]).
area([
        p(26.65,28.99), p(28.34,30.16), p(30.29,28.08),
        p(28.34,26.91), p(27.04,26.91), p(26.65,28.99)],
        [land_use(urban),density([60,66])]).
area([
        p(49.79,26.13), p(53.82,27.95), p(55.90,26.00),
        p(53.82,23.79), p(49.79,24.83), p(49.79,26.13)],
        [land_use(urban),density(55)]).

area([
        p(17.16,43.16), p(22.75,45.50), p(21.19,33.67),
        p(16.25,33.15), p(17.16,43.16)],
        [land_use(farmland),density(15)]).
area([
        p(32.63,45.11), p(37.70,46.02), p(37.70,46.02),
        p(34.58,35.88), p(26.13,34.58), p(32.63,45.11)],
        [land_use(farmland),density(18)]).
area([
        p(14.04,21.84), p(22.88,26.65), p(26.13,26.65),
        p(23.14,20.67), p(14.04,21.84)],
        [land_use(farmland),density(22)]).
area([
        p(46.41,24.18), p(53.43,21.97), p(55.38,16.25),
        p(47.97,17.16), p(46.41,24.18)],
        [land_use(farmland),density([13,18])]).
area([
        p(51.61,30.42), p(56.68,32.89), p(55.64,37.83),
        p(58.63,38.74), p(60.84,30.68), p(56.81,27.04),
        p(51.61,30.42)],
        [land_use(farmland),density(18)]).
```

```
area([
        p( 0.00, 0.00), p( 0.00,47.32), p(61.36,45.24),
        p(62.27,13.65), p( 0.00, 0.00)],
        [land_use(wild),density(0)]).
```

/* ***Topography Terrain Model*** */

```
area([
        p( 1.43,41.99), p( 4.81,42.90), p( 5.72,42.25),
        p( 5.20,39.13), p( 1.95,39.13), p( 1.43,41.99)],
        [topography(rough)]).
area([
        p( 3.90,29.77), p( 5.98,28.99), p( 5.59,23.79),
        p( 3.64,22.88), p( 0.91,25.74), p( 3.90,29.77)],
        [topography(rough)]).
area([
        p( 8.97,39.13), p(15.08,34.84), p(13.52,32.24),
        p(10.66,30.81), p( 7.93,33.15), p( 8.97,39.13)],
        [topography(rough)]).
area([
        p(13.52,42.25), p(21.32,41.08), p(18.59,38.09),
        p(14.43,38.09), p(12.22,40.17), p(13.52,42.25)],
        [topography(rough)]).
area([
        p(23.40,39.13), p(26.39,40.04), p(27.43,37.31),
        p(24.31,34.06), p(25.61,32.11), p(22.23,31.07),
        p(19.24,33.02), p(23.40,39.13)],
        [topography(rough)]).
area([
        p(14.04,28.86), p(19.63,27.04), p(22.36,28.73),
        p(23.79,27.82), p(22.23,26.00), p(15.34,22.75),
        p(14.04,26.13), p(14.04,28.86)],
        [topography(rough)]).

area([
        p( 1.04,46.41), p(29.25,45.50), p(32.11,39.39),
        p(23.79,29.25), p(34.06,20.67), p(30.55,12.61),
        p(14.43,19.89), p( 3.77,22.88), p(-0.39,26.39),
        p( 0.39,28.99), p( 3.38,33.67), p( 0.65,32.11),
        p( 1.04,46.41)],
        [topography(rolling)]).
area([
        p(39.52,41.34), p(48.75,43.16), p(57.98,37.18),
        p(56.68,28.99), p(53.43,26.91), p(49.40,21.84),
        p(38.61,22.88), p(35.49,33.15), p(40.43,35.23),
        p(39.52,41.34)],
        [topography(rolling)]).
```

```
area([
        p(-0.52,24.83), p( 3.38,21.84), p(15.86,17.81),
        p(12.22,17.81), p(14.04,15.73), p(18.20,14.82),
        p(18.98,15.47), p(22.23,15.73), p(27.30, 9.49),
        p(26.39, 8.58), p(24.18,11.57), p(13.00,14.56),
        p( 2.34,20.93), p(-0.39,21.84), p(-0.52,24.83)],
        [topography(waterbody)]).

area([
        p( 0.00, 0.00), p( 0.00,47.32), p(61.36,45.24),
        p(62.27,13.65), p( 0.00, 0.00)],
        [topography(hilly)]).

/* Linear Entities */
highway([],[
        p(37.31,46.15), p(34.32,33.15), p(27.04,31.07),
        p(25.09,27.17), p(21.97,28.34), p(12.61,24.83),
        p(11.70,19.63), p( 4.55, 9.49)]).

highway([],[
        p(34.32,33.15), p(35.36,28.08), p(34.19,28.08),
        p(38.48,19.89), p(38.22,15.73), p(34.97,10.40)]).

highway([],[
        p(35.36,28.08), p(38.61,21.97), p(42.38,21.97),
        p(46.93,16.90), p(42.64,11.83)]).

/* These are routes to be analyzed in Chapter 5 */

chap5 :-

    analyze_route(2,
                [p( 5,30),p( 9,27),p(12,27),p(14,30),p(25,28)]
                    ),

    analyze_route(2,
                [p(25,28),p(40,33),p(49,29),p(54,23),p(58,22)]
                    ),

    analyze_route(2,
                [p(25,28),p(33,27),p(42,21),p(50,19),p(54,23),p(58,22)]
                    ),

    analyze_route(2,
                [p(25,28),p(37,19),p(44,17),p(50,19),p(54,23),p(58,22)]
                    ).
```

```
search__area(
                [p(24,15),p(24,30),p(58,30),p(58,15),p(24,15)]
                    ).
```

Appendix 2. Sample Interaction with "WHY?" Explanations

```
?- solve(evaluate_attribute(p(5,30),Impact).

>What is the adt (value,why,unknown) ? why.
+-----------------------------------------------
| air_quality_impact(p(5,30),_0241)
| Can be shown using the following rule:
+-----------------------------------------------
| air_quality_impact(p(5,30),_0241) IS TRUE IF
|    adt(_043D) AND
|    design_speed(_0451) AND
|    emission_level(_043D,_0451,_046D) AND
|    region_data(p(5,30),land_use(_048D)) AND
|    sensibility_to_pollution(_048D,_04B1) AND
|    region_data(p(5,30),density(_04D1)) AND
|    fuzzy_add([_04D1,_04B1,_046D],_0241)
+-----------------------------------------------

>What is the adt (value,why,unknown) ? why.
+-----------------------------------------------
| evaluate_attribute(p(5,30),physical(_0221))
| Can be shown using the following rule:
+-----------------------------------------------
| evaluate_attribute(p(5,30),physical(_0221))
| IS TRUE IF
|    air_quality_impact(p(5,30),_0241) AND
|    water_quality_impact(p(5,30),_0241) AND
|    noise_impact(p(5,30),_0271) AND
|    fuzzy_add([_0241,_0259,_0271],_0221)
+-----------------------------------------------

>What is the adt (value,why,unknown) ? why.
*** Sorry, there is no further explanation.

>What is the adt (value,why,unknown) ? unknown.
*** In this case, I'll assume adt([10000,12000])...

>What is the design_speed (value,why,unknown) ? 100.

>What is the truck_percentage (value,why,unknown) ? unknown.
*** In this case, I'll assume truck_percentage([0.12,0.01,0.01])...

Impact = physical([7705.0,700.0,700.0]) ->;
```

```
Impact = aesthetic([135.386,12.063,12.063]) ->;

Impact = relocation(0) ->;

Impact = construction([628.5,48.5,48.5]) ->;

>What is the number_of_lanes (value,why,unknown) ? why.
+-----------------------------------------------
| accident_costs(_0169)
| Can be shown using the following rule:
+-----------------------------------------------
| accident_costs(_0169) IS TRUE IF
|    adt([10000,12000]) AND
|    number_of_lanes(_626D) AND
|    yearly_accident_cost(_626D,[10000,12000],_0169)
+-----------------------------------------------

>What is the number_of_lanes (value,why,unknown) ? 4.

Impact = operational([7097.6190,1767.1182,1767.1182]) ->;

no (this indicates there are no further solutions)
```

Appendix 3. Sample Interaction with "HOW?" Explanations

```
?- how(evaluate_point(p(28.21,29.07),J,K)).

Please wait while I think about
evaluate_point(p(28.21,29.07),_0085,_0095)...

*** Yes, I can show that
evaluate_point(p(28.21,29.07),[physical([0.93,0.07,0.07]),
                                aesthetic([0.92,0.08,0.08]),
                                relocation([0.63,0.03,0.03]),
                                construction([0.68,0.05,0.05]),
                                operational([0.69,0.17,0.17])],
                               [3.66,0.39,0.45]).

>Should I explain this further? y.
+-----------------------------------------------
| evaluate_point(p(28.21,29.07),[physical([0.93,0.07,0.07]),
|                                 aesthetic([0.92,0.08,0.08]),
|                                 relocation([0.63,0.03,0.03]),
|                                 construction([0.68,0.05,0.05]),
|                                 operational([0.69,0.17,0.17])],
|                                [3.66,0.39,0.45])
| Can be shown using the following rule:
+-----------------------------------------------
| evaluate_point(p(28.21,29.07),[physical([0.93,0.07,0.07]),
|                                 aesthetic([0.92,0.08,0.08]),
|                                 relocation([0.63,0.03,0.03]),
|                                 construction([0.68,0.05,0.05]),
|                                 operational([0.69,0.17,0.17])],
|                                [3.66,0.39,0.45])
| IS TRUE IF
|     findall(_0239,evaluate_attribute(p(28.21,29.07),
|                       [physical([12369.67,921.89,921.89]),
|                         aesthetic([135.57,11.34,11.34]),
|                         relocation([60,66]),
|                         construction([1514.5,116.5,116.5]),
|                         operational([7727.67,1909.37,1909.37])) AND
|     scale_list([physical([12369.67,921.89,921.89]),
|                         aesthetic([135.57,11.34,11.34]),
|                         relocation([60,66]),
|                         construction([1514.5,116.5,116.5]),
|                         operational([7727.67,1909.37,1909.37])],
|                       [physical([0.93,0.07,0.07]),
|                         aesthetic([0.92,0.08,0.08]),
```

```
|                          relocation([0.63,0.03,0.03]),
|                          construction([0.68,0.05,0.05]),
|                          operational([0.69,0.17,0.17])]) AND
|      aggregate_impacts([physical([0.93,0.07,0.07]),
|                                   aesthetic([0.92,0.08,0.08]),
|                                   relocation([0.63,0.03,0.03]),
|                                   construction([0.68,0.05,0.05]),
|                                   operational([0.69,0.17,0.17])],
|                                  [3.66,0.39,0.45]).
+-----------------------------------------------

evaluate_attribute(p(28.21,29.07),physical([12369.7,921.9,921.9]))

>Should I explain this further? y.
+-----------------------------------------------
| evaluate_attribute(p(28.21,29.07),
|                                  physical([12369.7,921.9,921.9]))
| Can be shown using the following rule:
+-----------------------------------------------
| evaluate_attribute(p(28.21,29.07),
|                                  physical([12369.7,921.9,921.9]))
| IS TRUE IF
|      air_quality_impact(p(28.21,29.07),[7773,703,703]) AND
|      water_quality_impact(p(28.21,29.07),0) AND
|      noise_impact(p(28.21,29.07),[4596.67,218.89,218.89]) AND
|      fuzzy_add([[7773,703,703],0,[4596.67,218.89,218.89]],
|                          [12369.67,921.89,921.89])
+-----------------------------------------------

air_quality_impact(p(28.21,29.07),[7773,703,703])

>Should I explain this further? y.
+-----------------------------------------------
| air_quality_impact(p(28.21,29.07),[7773,703,703])
| Can be shown using the following rule:
+-----------------------------------------------
| air_quality_impact(p(28.21,29.07),[7773,703,703])
| IS TRUE IF
|      adt([10000,12000]) AND
|      design_speed(100) AND
|      emission_level([10000,12000],100,[7700,700,700]) AND
|      region_data(p(28.21,29.07),land_use(urban)) AND
|      sensibility_to_pollution(urban,10) AND
|      region_data(p(28.21,29.07),density([60,66]) AND
|      fuzzy_add([[60,66],10,[7700,700,700]],[7773,703,703])
+-----------------------------------------------
```

```
adt([10000,12000])

>Should I explain this further? y.
+-----------------------------------------------
| adt([10000,12000])
| Is a fact from the database
+-----------------------------------------------

design_speed(100)

>Should I explain this further? y.
+-----------------------------------------------
| design_speed(100)
| Is a fact from the database
+-----------------------------------------------

emission_level([10000,12000],100,[7700,700,700])

>Should I explain this further? y.
+-----------------------------------------------
| emission_level([10000,12000],100,[7700,700,700])
| Can be shown using the following rule:
+-----------------------------------------------
| emission_level([10000,12000],100,[7700,700,700])
| IS TRUE IF
|      fuzzify(100,[100,0,0]) AND
|         100 < 80 AND
|         [7700,700,700] is [10000,12000] OR
|      fuzzy_multiply([10000,12000],0.7,[7700,700,700])
+-----------------------------------------------

fuzzify(100,[100,0,0])

>Should I explain this further? n.

fuzzy_multiply([10000,12000],0.7,[7700,700,700])

>Should I explain this further? y.
+-----------------------------------------------
| fuzzy_multiply([10000,12000],0.7,[7700,700,700])
| See Prade & Dubois, "Operations on Fuzzy Numbers"
+-----------------------------------------------

region_data(p(28.21,29.07),land_use(urban))

>Should I explain this further? n.
```

```
sensibility_to_pollution(urban,10)

>Should I explain this further? n.

region_data(p(28.21,29.07),density([60,66]))

>Should I explain this further? n.

fuzzy_add([[60,66],10,[7700,700,700]],[7773,703,703]))

>Should I explain this further? n.

water_quality_impact(p(28.21,29.07),0)

>Shouid I explain this further? y.
+-----------------------------------------------
| water_quality_impact(p(28.21,29.07),0)
| Can be shown using the following rule:
+-----------------------------------------------
| water_quality_impact(p(28.21,29.07),0)
| IS TRUE IF
|      not region_data(p(28.21,29.07),topography(waterbody))
+-----------------------------------------------

noise_impact(p(28.21,29.07),[4596.67,218.89,218.89])

>Should I explain this further? y.
+-----------------------------------------------
| noise_impact(p(28.21,29.07),[4596.67,218.89,218.89])
| Can be shown using the following rule:
+-----------------------------------------------
| noise_impact(p(28.21,29.07),[4596.67,218.89,218.89])
| IS TRUE IF
|      adt([10000,12000]) AND
|      fuzzy_divide([10000,12000],24,[458.33,41.67,41.67]) AND
|      design_speed(100) AND
|      fuzzify([100,[100,0,0]) AND
|      truck_percentage([0.12,0.01,0.01]) AND
|      fuzzify([0.12,0.01,0.01],[0.12,0.01,0.01]) AND
|      145 is 100+40+500/100 AND
|      1.006 is 1+5*0.12/100 AND
|      72.96 is 10 * log(458.33)+33*log(145)+10*log(1.006)-25 AND
|      region_data(p(28.21,29.07),density([60,66]) AND
|         72.96 < 50 AND
|         fuzzy_multiply(50,[60,66],[4596.67,218.89,218.89]) OR
|         72.96 > 85 AND
|         fuzzy_multiply(85,[60,66],[4596.67,218.89,218.89]) OR
```

```
|      fuzzy__multiply(72.96,[60,66],[4596.67,218.89,218.89])
+-----------------------------------------------
```

adt([10000,12000])

>Should I explain this further? ***n.***

fuzzy__divide([10000,12000],24,[458.33,41.67,41.67])

>Should I explain this further? ***n.***

design__speed(100)

>Should I explain this further? ***n.***

fuzzify([100,[100,0,0])

>Should I explain this further? ***n.***

truck__percentage([0.12,0.01,0.01])

>Should I explain this further? ***n.***

fuzzify([0.12,0.01,0.01],[0.12,0.01,0.01])

>Should I explain this further? ***n.***

region__data(p(28.21,29.07),density([60,66])

>Should I explain this further? ***n.***

********** etc... **********

?- ***how(evaluate__attribute(p(28.21,29.07),operational(O))).***

Please wait while I think about
evaluate__attribute(p(28.21,29.07),operational(O))

*** Yes, I can show that
evaluate__attribute(p(28.21,29.07),operational(O))

>Should I explain this further? ***y.***

```
+-----------------------------------------------
| evaluate__attribute(p(28.21,29.07),operational(O))
| Can be shown using the following rule:
+-----------------------------------------------
```

```
| evaluate_attribute(p(28.21,29.07),operational(O))
| IS TRUE IF
|    average_speed(car,100,[95,5,5]) AND
|    speed_factor(car,[95,5,5],[0.0883,0.0068,0.0068]) AND
|    region_data(p(28.21,29.07),topography(hilly) AND
|    grade_factor(car,hilly,[1.13,1.2]) AND
|    fuzzy_multiply([0.088,0.007,0.007],[1.13,1.2],[0.1,0.01,0.01])
+-----------------------------------------------

average_speed(car,100,[95,5,5])

>Should I explain this further? y.
+-----------------------------------------------
| average_speed(car,100,[95,5,5])
| Can be shown using the following rule:
+-----------------------------------------------
| average_speed(car,100,[95,5,5])
| IS TRUE IF
|      fuzzy_multiply([100,0.9,1],[95,0.5,0.5])
+-----------------------------------------------

fuzzy_multiply([100,0.9,1],[95,0.5,0.5])

>Should I explain this further? y.
+-----------------------------------------------
| fuzzy_multiply([100,0.9,1],[95,0.5,0.5])
| See Prade & Dubois, "Operations on Fuzzy Numbers"
+-----------------------------------------------

      ************************** etc... **************************

travel_time_monetary_costs([163.81,40.91,40.91])

>Should I explain this further? y.
+-----------------------------------------------
| travel_time_monetary_costs([163.81,40.91,40.91])
| Can be shown using the following rule:
+-----------------------------------------------
| travel_time_monetary_costs([163.81,40.91,40.91])
| IS TRUE IF
|      design_speed(100) AND
|      average_speed(truck,100,[50,5,5]) AND
|      driver_wage(12) AND
|      adt([10000,12000]) AND
|      truck_percentage([0.12,0.01,0.01]) AND
|      commercial_car_travel(0.05) AND
|      yearly_time_cost([0.12,0.01,0.01],0.05,[10000,12000],
```

```
|                                        [50,5,5],12,[163.81,40.91,40.91])
+-----------------------------------------
```

design__speed(100)

>Should I explain this further? ***n.***

average__speed(truck,100,[50,5,5])

>Should I explain this further? ***n.***

driver__wage(12)

>Should I explain this further? ***y.***

```
+-----------------------------------------
| driver__wage(12)
| Is a fact from the database
+-----------------------------------------
```

adt([10000,12000])

>Should I explain this further? ***n.***

truck__percentage([0.12,0.01,0.01])

>Should I explain this further? ***n.***

commercial__car__travel(0.05)

>Should I explain this further? ***y.***

```
+-----------------------------------------
| commercial__car__travel(0.05)
| Is a fact from the database
+-----------------------------------------
```

>Should I explain this further? ***n.***

********** etc... **********

Appendix 4. Partial Program Listing

*** NOTICE: ***
This partial listing of HLA's code is provided to illustrate the ideas and principles involved in this project. However, HLA and its code are proprietary and copyrighted by the author and by the University of British Columbia. Reproduction in whole or in part for non-educational purposes without the author's written permission is unlawful.

UTILITY MODULE

```
/*
Module UTILITY
Util.Ari
================================================
You guessed it. This file is full of utilities, some written
by me, others taken from C&M, Turbo & Arity manuals etc.
================================================
Bernardo de Castilho, nov.86
*/

/*-------------------------------------------------
common list utilities
*/

append([],L,L).
append([H|T],L,[H|T1]) :-
    append(T,L,T1).

member(H,[H|_]).
member(H,[_|T]) :-
    member(H,T).

/*
    This predicate merges two lists adding term values and without
replicating functors.
for example:
merge_list([e(4),p(2)],[r(2),t(7),e(5),p(4)],[r(2),t(7),e(9),p(6)])
*/
merge_list(L1,[],L1) :-  !.
merge_list(L1,[Head|Tail],NL) :-
    merge_elem(L1,Head,L2),
    merge_list(L2,Tail,NL).
```

```
merge_elem([],Elem,[Elem]) :-   !.
merge_elem([Head|Tail],Elem,L) :-
    Head =.. [F,Hv],
    Elem =.. [F,Ev],
    fuzzy_add(Hv,Ev,V),
    Nelem =.. [F,V],
    L = [Nelem|Tail], !.
merge_elem([Head|Tail],Elem,L) :-
    merge_elem(Tail,Elem,NL),
    L = [Head|NL], !.

/*-----------------------------------------------
Some output utilities:
*/
print([]) :-   !.
print([tab(X)|T]) :-
    !,
    tab(X),
    print(T).
print([nl|T]) :-
    !,
    nl,
    print(T).
print([H|T]) :-
    !,
    write(H),
    print(T).

display_rule(rule(Goal,Body)) :-
    upper_line,
    print([nl,$|  $,Goal,nl,
              $|  Can be shown using the following rule:$,nl]),
    middle_line,
    print([$|  $,Goal,$  IS TRUE IF$,nl]),
    display_body(3,Body),
    lower_line, !.

display_body(N,([!X!])) :- /* first get rid of snips */
    display_body(N,X), !.
display_body(N,(X,!)) :-
    display_body(N,X), !. /* and cuts */
display_body(N,(!,X)) :-
    display_body(N,X), !.
display_body(N,(X)) :-
    (
        X = (Head,(Tail)),
        not compound(Head),
```

```
            S = $ AND$
        ;
            X = (Head;(Tail)),
            not compound(Head),
            S = $ OR$
        ),
        !,
        display_body(N,Head),
        print([S,nl]),
        display_body(N,Tail).
display_body(N,(X)) :-
        (
            X = (Head,(Tail)),
            S = $ AND$
        ;
            X = (Head;(Tail)),
            S = $ OR$
        ),
        N1 is N+3,
        display_body(N1,Head),
        print([S,nl]),
        display_body(N,Tail).
display_body(N,X) :-
        print([$|$,tab(N),X]), !.

compound((_,_)) :- !.
compound((_;_)) :- !.

upper_line :-
        write($V$),
        wc(79,+D).
middle_line :-
        write($G$),
        wc(79,+D),
        nl.
lower_line :-
        nl,
        write($S$),
        wc(79,+D),
        nl.

/*-----------------------------------------
miscellaneous
*/
for(X,X,X) :- !.
for(Y,Y,X).
for(Z,X,Y) :-
```

```
    inc(X,X1),
    for(Z,X1,Y).

odd(X,L) :-
    count(X,L,C),
    1 is C mod 2.

count(_,[],0) :-  !.
count(X,[X|R],C) :-
    !,
    count(X,R,C1),
    inc(C1,C).
count(X,[_|R],C) :-
    !,
    count(X,R,C).

retractall(X) :-
    retract(X),
    fail.
retractall(_) :-  !.

/*
Geom.Ari
=========================================================
This file contains some Prolog predicates related to geometry.
The main entry points are
    1. intercept(P1,P2,P3,P4):
    This call would succeed if the straight line segment from P1 to
P2 and the one from P3 to P4 crossed each other. Useful for
detecting interactions with other linear entities such as roadways,
cattle crossings etc.
    2. in_area(Point,Area):
    This predicate succeeds if 'Point' lies inside the area
represented by a sequence of vertices. Areas can be ANY reasonable
shape. Concave, convex, crossing lines, ok.
    3. distance(P1,P2,Distance):
    This is pretty obvious. Called by the other predicates and useful
for detecting interactions with point entities such as hospitals,
schools and the like.
---------------------------------------------------------
written in Nov.86 by Bernardo de Castilho
(13.11 update: convex area membership supported)
(17.11 update: ported from Turbo)
(20.11 update: intercept predicates added)
=========================================================
*/
```

```
/*-------------------------------------------------
these are called by the route analyzer predicate
*/
get_incr(StepSize,p(X1,Y1),p(X2,Y2),
                Spacing,Npoints,Xincr,Yincr) :-
    distance(p(X1,Y1),p(X2,Y2),Dist),
    Dist>0,
    get_n_points(Dist,StepSize,Npoints),
    Spacing is Dist/Npoints,
    Xincr is (X2-X1)/Npoints,
    Yincr is (Y2-Y1)/Npoints.

get_n_points(Dist,StepSize,Npoints) :-
    Npoints is integer(Dist/StepSize),
    Npoints>0,
    !.
get_n_points(_,_,1) :-  !.

get_segment(p(XC,YC),Xincr,Yincr,p(X1,Y1),p(X2,Y2)) :-
    X1 is XC-Xincr/2,
    X2 is XC+Xincr/2,
    Y1 is YC-Yincr/2,
    Y2 is YC+Yincr/2, !.

/*-------------------------------------------------
succeeds if the two straight line segments intercept
*/
intercept(p(X1,Y1),p(X2,Y2),p(X3,Y3),p(X4,Y4)) :-
    get_crossing_point(p(X1,Y1),p(X2,Y2),p(X3,Y3),p(X4,Y4),p(X,Y)),
    between(X,X1,X2),
    between(X,X3,X4),
    between(Y,Y1,Y2),
    between(Y,Y3,Y4),
    !.

get_crossing_point(p(X,_),p(X,_),p(X3,Y3),p(X4,Y4),p(X,Y)) :-
    get_angle(p(X3,Y3),p(X4,Y4),Beta),
    Y is Beta * (X-X3) + Y3, !.
get_crossing_point(p(X1,Y1),p(X2,Y2),p(X3,Y3),p(X4,Y4),p(X,Y)) :-
    get_angle(p(X1,Y1),p(X2,Y2),Alpha),
    get_angle(p(X3,Y3),p(X4,Y4),Beta),
    Alpha /= Beta,
    X is (Y3-Y1 + X1*Alpha - X3*Beta) / (Alpha-Beta),
    Y is Alpha * (X-X1) + Y1, !.

get_angle(p(X,_),p(X,_),9.9999E99) :-  !.
get_angle(p(X1,Y1),p(X2,Y2),Angle) :-
```

```
    Angle is (Y2-Y1) / (X2-X1), !.

/*----------------------------------------
succeeds if Point is inside the area defined by the polygon
called Area. The point is inside if its relative position with
respect to any sides of the polygon is:
   a) above, below, left AND right of any segment OR
   b) or ON any segment.
*/
in_area(Point,Area) :-
    make_rel_pos(Point,Area,[],Rel_Pos),
    (
      member(on,Rel_Pos)
    ;
      odd(a,Rel_Pos),
      odd(b,Rel_Pos),
      odd(l,Rel_Pos),
      odd(r,Rel_Pos)
    ), !.

/*
    create a list of positions of the point relative to each line
segment: [a] is above, [b] is below, [a,b] is on the segment.
*/
make_rel_pos(_,[_],X,X) :-  !.     /* one point left, stop. */
make_rel_pos(Point,[P1,P2|Rest],Old,Rel_Pos) :-
    rel_pos(Point,P1,P2,Pos),
    append(Old,Pos,New),
    make_rel_pos(Point,[P2|Rest],New,Rel_Pos).

rel_pos(Point,Start,End,P) :-
    get_v_stat(Point,Start,End,V),
    get_h_stat(Point,Start,End,H),
    append(V,H,P), !.

get_v_stat(p(X,Y),p(X1,Y1),p(X2,Y2),V) :-
    between(X,X1,X2),
    above_or_below(p(X,Y),p(X1,Y1),p(X2,Y2),V), !.
get_v_stat(_,_,_,[]) :-  !.

get_h_stat(p(X,Y),p(X1,Y1),p(X2,Y2),H) :-
    between(Y,Y1,Y2),
    left_or_right(p(X,Y),p(X1,Y1),p(X2,Y2),H), !.
get_h_stat(_,_,_,[]) :-  !.

above_or_below(p(_,Y),p(_,Y1),p(_,Y2),[a]) :-
    Y > Y1,
```

```
    Y > Y2, !.
above_or_below(p(_,Y),p(_,Y1),p(_,Y2),[b]) :-
    Y < Y1,
    Y < Y2, !.
above_or_below(_,p(X1,_),p(X2,_),[a,b]) :-
    X1 = X2, !.
above_or_below(p(X,Y),p(X1,Y1),p(X2,Y2),V) :-
    DP is Y-Y1,
    DL is (X-X1) * (Y2-Y1) / (X2-X1),
    compare_v(DP,DL,V), !.

compare_v(DP,DL,[a]) :-
    DP>DL, !.
compare_v(DP,DL,[b]) :-
    DP < DL, !.
compare_v(_,_,[on]) :-  !.

left_or_right(p(X,_),p(X1,_),p(X2,_),[r]) :-
    X > X1,
    X > X2, !.
left_or_right(p(X,_),p(X1,_),p(X2,_),[l]) :-
    X < X1,
    X < X2, !.
left_or_right(_,p(_,Y1),p(_,Y2),[l,r]) :-
    Y1 = Y2, !.
left_or_right(p(X,Y),p(X1,Y1),p(X2,Y2),H) :-
    DP is X-X1,
    DL is (Y-Y1) * (X2-X1) / (Y2-Y1),
    compare_h(DP,DL,H), !.

compare_h(DP,DL,[r]) :-
    DP>DL, !.
compare_h(DP,DL,[l]) :-
    DP < DL, !.
compare_h(_,_,[on]) :-  !.

between(X,X1,X2) :-
    X1 > X, X >= X2;
    X2 >= X, X > X1, !.

distance(p(X1,Y1),p(X2,Y2),Dist) :-
    Dx is X2-X1,
    Dy is Y2-Y1,
    Dist is sqrt(Dx*Dx+Dy*Dy).

/*
```

Fuzzy.Ari

```
/*=====================================================
This file contains fuzzy number operators and clauses that convert
real numbers or ranges into fuzzy numbers.
(see Prade, H. and Dubois, D. (1978). "Operations on
fuzzy numbers". International Journal of Systems Science, Vol.9,
No.6, pp.613-626.)
=====================================================
written in nov.86 by Bernardo de Castilho
=====================================================
*/

/*-------------------------------------
This predicate convert reals or ranges into fuzzy numbers:
*/
fuzzify([M,A,B],[M,A,B]) :-   !.          /* already fuzzy */
fuzzify([L,H],[M,A,B]) :-                 /* range         */
     M is (L+H)/2,
     A is abs((H-L)/2),
     B is A, !.
fuzzify(M,[M,0,0]) :-   !.                /* real          */

/*-------------------------------------
These are some fuzzy number operators:
*/
fuzzy_add(M1,N1,[Rm,Ra,Rb]) :-
     fuzzify(M1,[M,A,B]),
     fuzzify(N1,[N,C,D]),
     Rm is M+N,
     Ra is A+C,
     Rb is B+D, !.

fuzzy_add([],0) :-   !.     /* this one adds up the terms of a list */
fuzzy_add([H|T],R) :-
     fuzzy_add(T,R1),
     fuzzy_add(H,R1,R), !.

fuzzy_subtract(M1,N1,[Rm,Ra,Rb]) :-
     fuzzify(M1,[M,A,B]),
     fuzzify(N1,[N,C,D]),
     Rm is M-N,
     Ra is A+C,
     Rb is B+D, !.

fuzzy_multiply(M1,N1,[Rm,Ra,Rb]) :-
     fuzzify(M1,[M,A,B]),
     fuzzify(N1,[N,C,D]),
     Rm is M*N,
```

```
    Ra is M*C+N*A,
    Rb is M*D+N*A, !.

fuzzy_divide(M1,N1,[Rm,Ra,Rb]) :-
    fuzzify(M1,[M,A,B]),
    fuzzify(N1,[N,C,D]),
    Rm is M/N,
    Ra is (M*D+N*A)/N/N,
    Rb is (M*D+N*B)/N/N, !.

fuzzy_power(M1,P1,Power) :-
    fuzzify(M1,[M,A,B]),
    fuzzify(P1,[N,C,D]),
    Lb is (M-A) ‡ (N-C),
    Ub is (M+B) ‡ (N+D),
    fuzzify([Lb,Ub],Power), !.

series_present_worth(N,I,Yearly_Cost,Cost) :-
    fuzzy_add(I,1,Iplus1),
    fuzzy_power(Iplus1,N,Iplus1_to_N),
    fuzzy_subtract(Iplus1_to_N,1,Over),
    fuzzy_multiply(Iplus1_to_N,I,Under),
    fuzzy_divide(Over,Under,P_A_N),
    fuzzy_multiply(P_A_N,Yearly_Cost,Cost),
    not member(err,Cost), !.          /* 'err' will appear if i=0 */
series_present_worth(N,_,Yearly_Cost,Cost) :-
    fuzzy_multiply(Yearly_Cost,N,Cost), !.
```

INFERENCE ENGINE MODULE

```
/*
Module INFERENG
IEhow.Ari
==========================================
Implements and extends the inference engine described in
"Expert System = Knowledge + Meta-Interpreter"
by Leon Sterling.
----------------------------------------------------
original Arity version written by Glen Cooper
further upgraded by Bernardo de Castilho.
   20.11: can handle disjunctions
   22.11: selective explanations
   27.11: can handle snipped conjunctions
   27.11: findall is ok (findally...)
   28.11: display_rule improved
            (now it can handle compound terms)
==========================================
*/

/*-----------------------------------------
Determine if a query can be proved.
Interpret its proof or state that no proof exists.
*/
how(Goal) :-
    print([$Please wait while I think about$,nl,Goal,$...$]),
    (
        prove(Goal,Proof),
        print([nl,$*** Yes, I can show that$,nl,Goal,nl]),
        interpret(Proof)
    ;
        print([nl,nl,$*** No, this seems to be either false or undefined.$,nl]),
        fail
    ), !.

/*-----------------------------------------
Prove a query, if possible, and save its proof along the way
*/
prove(true,true) :-  !.                    /* this is trivial */
prove((Goal1;Goal2),Proof) :-        /* this works with disjunctions */
    prove(Goal1,Proof);
    prove(Goal2,Proof).
prove((Goal1,Goal2),(Proof1,Proof2)) :-
    !,                                          /* this works with conjunctions */
    prove(Goal1,Proof1),
    prove(Goal2,Proof2).
prove([!(Goal1,Goal2)!],(Proof1,Proof2)) :-
```

```
    !,                                          /* this works with conjunctions */
    [!
        prove(Goal1,Proof1),
        prove(Goal2,Proof2)
    !].
prove(Goal,implies(Proof,Goal,true)) :-      /* fact in disguise... */
    clause(Goal,!).
prove(Goal,implies(Proof,Goal,obvious)) :- /* prove subgoals */
    obvious(Goal,_),
    call(Goal).                          /* prove but don't keep the proof */
prove(Goal,implies(Proof,Goal,Body)) :-      /* prove subgoals */
    not obvious(Goal,_),
    clause(Goal,Body),
    prove(Body,Proof).
prove(Goal,implies(system,Goal)) :-            /* built in predicates */
    functor(Goal,Name,Arity),
    system(Name/Arity),
    call(Goal), !.

/*----------------------------------------
Interpret a proof
*/
interpret(true) :-  !.                              /* this is trivial */
interpret([!(Proof)!]) :-
    !,
    interpret(Proof).
interpret((Proof1,Proof2)) :-
    !,
    interpret(Proof1),
    interpret(Proof2).
interpret(implies(true,Goal,true)) :-       /* if the body is true */
    !,                                                  /* then it's a fact      */
    print([nl,Goal]),
    (
        want_to_go_on,
        upper_line,
        print([nl,$| $,Goal,nl,$| Is a fact from the database.$]),
        lower_line
    ;
        true
    ).
interpret(implies(true,Goal,obvious)) :-
    !,
    print([nl,Goal]),
    (
        want_to_go_on,
        obvious(Goal,Explanation),
```

```
        upper_line,
        print([nl,$|  $,Goal,nl,$|  $,Explanation]),
        lower_line
    ;
        true
    ).
interpret(implies(Proof,Goal,Body)) :-
    !,                                  /* finally, goal with subgoals */
    print([nl,Goal]),
    (
        want_to_go_on,
        display_rule(rule(Goal,Body)),
        interpret(Proof)
    ;
        true
    ).
interpret(implies(system,findall(_,Goal,_))) :-
    !,
    (
        prove(Goal,Proof), /* to prove a 'findall' (or 'bagof' or 'setof') */
        interpret(Proof),   /* it's necessary to satisfy the goal as many    */
        fail                        /* times as possible.
*/
    ;
        true
    ).
interpret(implies(system,Goal)) :-   !. /* No need to explain built-ins */

/*----------------------------------------
housekeeping:
*/
want_to_go_on :-
    !,
    gc(full),
    print([nl,$Should I explain this further? $]),
    get0(A),
    nl,
    (
        A = †Y;
        A = †y
    ).
/*
IEwhy.Ari
==========================================
Extension of the Meta-Interpreter described in
"The art of Prolog",
by Leon Sterling and Ehud Shapiro, pp.315
```

```
------------------------------------------------
INTERACTIVE SHELL WITH USER QUERY AND 'WHY' EXPLANATIONS
================================================
*/

/*----------------------------------------
Solve a goal, prompt user for missing information and answer
eventual why questions.
*/
solve(Goal) :-    solve(Goal,[]).
solve(true,[]) :-    !.
solve(Goal,[]) :-
     clause(Goal,!).
solve((Goal1;Goal2),Rules) :-
     solve(Goal1,Rules);
     solve(Goal2,Rules).
solve((Goal1,Goal2),Rules) :-
     !,
     solve(Goal1,Rules),
     solve(Goal2,Rules).
solve([!(Goal1,Goal2)!],Rules) :-
     !,
     [!
          solve(Goal1,Rules),
          solve(Goal2,Rules)
     !].
solve(Goal,Rules) :-
     obvious(Goal,Explanation),
     !,
     call(Goal).
solve(Goal,Rules) :-
     clause(Goal,Body),
     solve(Body,[rule(Goal,Body)|Rules]).
solve(findall(X,Goal,List),Rules) :-
     (
          solve(Goal,Rules),
          fail
     )
     ;
     !,
     findall(X,Goal,List).
solve(findall(_,Goal,[]),Rules) :-    !.
solve(Goal,_) :-
     functor(Goal,Name,Arity),
     system(Name/Arity),
     !,
     call(Goal).
```

```
solve(Goal,Rules) :-        /* Everything else failed. Try user.*/
     not clause(Goal,_),
     askable(Goal),
     ask(Goal,Answer),
     respond(Answer,Goal,Rules).

/*
Prompt user for an answer. Acceptable ones are:
     1. The actual value -> accept it, go on
     2. A 'why' question, answer and ask again (recursion)
     3. User doesn't know. Use default.
*/
ask(Goal,Answer) :-
     Goal =.. [Functor|_],
     print([nl,$> What is the $,Functor,$ (value,why,unknown) ? $]),
     read(Answer).

respond(why,Goal,[Rule|Rules]) :-          /* answer 'why' */
     display_rule(Rule),
     ask(Goal,Answer),
     respond(Answer,Goal,Rules).
respond(why,Goal,[]) :-
     !,
     print([nl,$*** Sorry, there is no further explanation.$,nl]),
     ask(Goal,Answer),
     respond(Answer,Goal,Rules).
respond(unknown,Goal,Rules) :-              /* use default */
     !,
     default(Goal),
     assertz(Goal),
     print([$*** In this case, I'll assume $,Goal,$...$,nl]).
respond(Answer,New_Fact,Rules) :-           /* remember answer */
     !,
     New_Fact =.. [Functor,Answer],
     assert(New_Fact).

/* ------------------------------------------
Something is askable if there is a default value for it.
*/
askable(X) :-
     default(Y),
     X =.. [F|_],
     Y =.. [F|_], !.
```

KNOWLEDGE BASE MODULE

```
/*
Module KBASE
Eval.Ari
==========================================
This file contains cell evaluation clauses.
There are three main data domains:
    1. SYSTEM. These attributes are built-in. The user can temporarily
override the system's values by including them in the project
description file (*.PDF), which is reconsulted. Examples are
components lives, corridor width, interest rates and construction
costs.
    2. DEFAULT. These are built-in values that are used only when the
user explicitly tells the system s/he is unable to provide a
reasonable estimate. Examples are traffic levels, traffic mix (% trucks) and number
of lanes.
    3. USER. These attributes should be supplied by the user, usually
included in the PDF file. If they are not found, the inference engine
will ask the user for a reasonable fuzzy estimate. The user then gets
a chance to ask 'why'questions. If the user is unsure about the
value, the system will then use a built-in default (domain 2) which
will generally have high associated uncertainty.
    The way this works is really simple: the inference engine uses
SYSTEM and USER attributes -- they look exactly the same -- until
some piece of data can't be found. It then asks the user and expands
the database with the information supplied or with the DEFAULT
clause that couldn't be found before because it was hidden inside a
'default' clause. Simple, huh?
==========================================
written in Nov.86 by Bernardo de Castilho
==========================================
*/

/*----------------------------------------
Point analyzer predicate
*/
evaluate_point(Point,Scaled_Impact_List,Aggregated_Impact) :-
    findall(Impact,evaluate_attribute(Point,Impact),Raw_Impact_List),
    gc(full),
    scale_list(Raw_Impact_List,Scaled_Impact_List),
    aggregate_impacts(Scaled_Impact_List,Aggregated_Impact),
    gc(full), !.

/*----------------------------------------
Route analyzer predicates
*/
analyze_route(_,[_]) :-          /* one point left, stop. */
```

```
    findall(X,unit_impact(X),UIList),
    (
        retract(unit_impact(_)),
        fail
    ;
        true
    ),
    fuzzy_add(UIList,TCI),
    print([nl,$
    +-----------------------------------------------------
    |  *** Route Completed. The cumulative impact is :
    |     : $,TCI,$
    +-----------------------------------------------------$,
    nl]), !.
analyze_route(StepSize,[P1,P2|Rest]) :-
    print([nl,$ *** Analyzing Alignment $,P1,$ - $,P2,nl]),
    analyze_alignment(StepSize,P1,P2),
    print([nl,$ *** This alignment is ready.$]),
    analyze_route(StepSize,[P2|Rest]).

analyze_alignment(StepSize,P1,P2) :-
    get_incr(StepSize,P1,P2,Spacing,Npoints,Xincr,Yincr),
    for(N,1,Npoints),
       compute_point(P1,N,Xincr,Yincr,P),
       evaluate_unit(P,Spacing,Xincr,Yincr),
       gc(full),
    fail.
analyze_alignment(_,_,_) :-   !.

compute_point(p(X1,Y1),N,Xincr,Yincr,p(X,Y)) :-
    X is X1 + (N-0.5) * Xincr,
    Y is Y1 + (N-0.5) * Yincr, !.

evaluate_unit(CentralPoint,Length,Xincr,Yincr) :-
    print([nl,$* Central Point: $,CentralPoint,tab(5),
                  $* Length: $,Length,nl]),
    /* find all point impacts ----------*/
    findall(I,evaluate_attribute(CentralPoint,I),RPIL),
    gc(full),
    /* find all linear entity impacts ----------*/
    get_segment(CentralPoint,Xincr,Yincr,P1,P2),
    findall(Limp,evaluate_linear(P1,P2,Limp),RLIL),
    gc(full),
    /* put them all together ----------*/
    merge_list(RPIL,RLIL,RIL),
    scale_list(RIL,SIL),
    gc(full),
```

```
    /* aggregate ----------------------------------------*/
    aggregate_impacts(SIL,LAG),
    fuzzy_multiply(Length,LAG,Total),
    gc(full),
    print([$ central point impacts: $,SIL,nl]),
    print([$ *** Aggregated Impact: $,Total,nl]),
    assertz(unit_impact(Total)), !.

/*-----------------------------------------
This predicate aggregates impacts corresponding to different
attributes (Impact_List) according to system (or user-defined)
component value levels (weights).
The absolute aggregated value can be used for comparison purposes
-- the higher the number the worse.
*/
aggregate_impacts([],0) :-  !.
aggregate_impacts([Head|Tail],Aggregated_Impact) :-
    aggregate_impacts(Tail,Tail_Impact),
    Head =.. [Impact_Type,Impact_Value],
    relative_weight(Impact_Type,Weight),
    fuzzy_multiply(Impact_Value,Weight,Head_Impact),
    fuzzy_add(Head_Impact,Tail_Impact,Aggregated_Impact).

/* ----------------------------------------
* This is the heart of the system.
* These predicates evaluate points according to several different
  attributes.
* They return a term of the form 'type(impact)' where impact is
  a fuzzy value ranging from 0 (best) to 1 (worst).
*/

/*-----------------------------------------
I.Environmental Attributes
*/
evaluate_attribute(Point,physical(Physical_Impact)) :-
    [!
        air_quality_impact(Point,AirImp),
        water_quality_impact(Point,WaterImp),
        noise_impact(Point,NoiseImp),
        fuzzy_add([AirImp,WaterImp,NoiseImp],Physical_Impact)
    !].
evaluate_attribute(Point,aesthetic(VisualImp)) :-
    [!
        adt(ADT),
        truck_percentage(TP),
        visual_context(Point,C),
        fuzzy_multiply(0.012,ADT,ADT12),
```

```
        fuzzy_multiply(1.3,TP,TP13),
        fuzzy_add([3.58,ADT12,TP13],I1),
        fuzzy_subtract(I1,C,VisualImp)
    !].

/*-----------------------------------------
II. Social Attributes
*/
evaluate_attribute(Point,relocation(Imp)) :-
    region_data(Point,density(Imp)).

/*-----------------------------------------
III.Economic Attributes
*/
/* 1.Construction Cost */
evaluate_attribute(Point,construction(Constr_Cost)) :-
    [!
      region_data(Point,land_use(Use)), /* land acquisition */
      land_cost(Use,PW_Land),
      region_data(Point,topography(Topo)),     /* b.earthwork */
      earthwork_cost(Topo,PW_EarthWork),
      region_data(Point,geology(Geo)),          /* c.foundation */
      foundation_cost(Geo,PW_Foundation),
      fuzzy_add([PW_Land,PW_EarthWork,PW_Foundation],Constr_Cost)
    !].

/* 2.Operational Cost */
evaluate_attribute(Point,operational(Oper_Cost)) :-
    [!
        vehicle_operation_costs(Point,YVOC),
        travel_time_monetary_costs(YTTC),
        accident_costs(YAC),
        fuzzy_add([YVOC,YTTC,YAC],Yearly_Oper_Cost),
        project_life(Life),
        interest_rate(I),
        series_present_worth(Life,I,Yearly_Oper_Cost,Oper_Cost)
    !].

/*-----------------------------------------
Linear Entity Impacts
*/
evaluate_linear(P1,P2,construction(E)) :-
    [!
        (
            crosses(P1,P2,highway,_),
            interchange_cost(Int)
        ;
```

```
            Int is 0
        ),
        (
            crosses(P1,P2,cattle_crossing,_),
            underpass_cost(Upc),
            extra_cowboy_cost(Ecc)
        ;
            Upc is 0,
            Ecc is 0
        ),
        fuzzy_add([Int,Upc,Ecc],TE),
        distance(P1,P2,D),
        fuzzy_divide(TE,D,E)
    !].

evaluate_linear(P1,P2,biotic(E)) :-
    [!
        (
            crosses(P1,P2,wildlife_crossing,[Sp,Vol]),
            species_importance(Sp,Imp),
            fuzzy_multiply(Vol,Imp,TE)
        ;
            E is 0
        ),
        distance(P1,P2,D),
        fuzzy_divide(TE,D,E)
    !].

/*----------------------------------------
Subpredicates used by evaluate_whatever
*/
/*
"crosses" is used by the linear entity evaluation predicates
and calls get_pair
*/
crosses(P1,P2,What,Attribs) :-
    A =.. [What,Attribs,Linear],
    !,
    call(A),
    get_pair(Linear,P3,P4),
    intercept(P1,P2,P3,P4), !.

get_pair([P1,P2|_],P1,P2).
get_pair([/T],P1,P2) :-
    get_pair(T,P1,P2).

/*
```

```
/* environmental impact assessment subpredicates:
*/
air_quality_impact(Point,A_Q_I) :-
    adt(ADT),
    design_speed(Speed),
    emission_level(ADT,Speed,EL),
    region_data(Point,land_use(Cover)),
    !,
    sensibility_to_pollution(Cover,Reg_Sens),
    region_data(Point,density(NofPeople)), !,
    fuzzy_add([NofPeople,Reg_Sens,EL],A_Q_I), !.

emission_level(ADT,Speed,EL) :-
    fuzzify(Speed,[S,_,_]),
    (
        S < 80,
        EL is ADT
    ;
        fuzzy_multiply(ADT,0.7,EL)
    ), !.

water_quality_impact(Point,0) :-
    not region_data(Point,topography(waterbody)), !.
water_quality_impact(Point,W_Q_I) :-
    region_data(Point,waterbody_use(WB_use)),
    !,
    yearly_trucks_with_hazardous_loads(YTHL),
    number_of_lanes(Lanes),
    accident_rate(Lanes,Acc_R),
    fuzzy_multiply(YTHL,Acc_R,Nacc),
    waterbody_importance(WB_use,WB_importance),
    !,
    fuzzy_multiply(Nacc,WB_importance,W_Q_I), !.

noise_impact(Point,Imp) :-
    adt(ADT),
    fuzzy_divide(ADT,24,Vph),        /* vehicles per hour */
    fuzzify(Vph,[Q,_,_]),
    design_speed(Dspeed),
    fuzzify(Dspeed,[V,_,_]),
    truck_percentage(TP),
    fuzzify(TP,[P,_,_]),
    Aux1 is V + 40 + 500/V,
    Aux2 is 1 + 5*P/V,
    Level is 10 * log(Q) + 33 * log(Aux1) + 10 * log(Aux2) - 25,
    region_data(Point,density(NofPeople)),
    (
```

```
        Level < 50,
        fuzzy_multiply(50,NofPeople,Imp)
    ;
        Level > 85,
        fuzzy_multiply(85,NofPeople,Imp)
    ;
        fuzzy_multiply(Level,NofPeople,Imp)
    ).

yearly_trucks_with_hazardous_loads(YTHL) :-
    adt(ADT),
    truck_percentage(TP),
    fuzzy_multiply(ADT,TP,NT),
    trucks_with_hazardous_loads(THL),
    fuzzy_multiply(NT,THL,DTHL),
    fuzzy_multiply(DTHL,365,YTHL), !.

/*
    economic impact assessment subpredicates:
*/
vehicle_operation_costs(Point,YVOC) :-
    design_speed(DSpeed),
    region_data(Point,topography(Topo)),
    car_operating_cost(DSpeed,Topo,COC),
    truck_operating_cost(DSpeed,Topo,TOC),
    adt(ADT),
    truck_percentage(TPerc),
    fuzzy_subtract(1,TPerc,CPerc),
    yearly_operating_cost(ADT,CPerc,COC,Yearly_COC),
    yearly_operating_cost(ADT,TPerc,TOC,Yearly_TOC),
    fuzzy_add(Yearly_COC,Yearly_TOC,YVOC), !.

travel_time_monetary_costs(TTMC) :-
    design_speed(DSpeed),
    average_speed(truck,DSpeed,Speed),
    driver_wage(Wage),
    adt(ADT),
    truck_percentage(TPerc),
    commercial_car_travel(CCT),
    yearly_time_cost(TPerc,CCT,ADT,Speed,Wage,TTMC), !.

accident_costs(YAC) :-
    adt(ADT),
    number_of_lanes(Lanes),
    yearly_accident_cost(Lanes,ADT,YAC), !.

yearly_accident_cost(Lanes,ADT,Yearly_Accident_Cost) :-
```

```
    accident_cost_factor(Lanes,ACF),
    fuzzy_multiply(ADT,365,Yearly_Total),
    fuzzy_divide(Yearly_Total,ACF,YACD),          /* yearly cost ($) */
    fuzzy_divide(YACD,1000,Yearly_Accident_Cost), !. /* ($1,000) */

accident_rate(Lanes,[0.869,1.600]) :-   Lanes =< 2, !.
accident_rate(_,[0.781,2.806]) :-   !.

yearly_operating_cost(ADT,Perc,OC,Yearly_OC) :-
    fuzzy_multiply(ADT,365,Yearly_Total),                  /* yearly traffic */
    fuzzy_multiply(Yearly_Total,Perc,Yearly_Volume),
    fuzzy_multiply(Yearly_Volume,OC,YOCD),                 /* yearly cost ($) */
    fuzzy_divide(YOCD,1000,Yearly_OC), !.                  /* ($1,000) */

car_operating_cost(Design_Speed,Topo,COC) :-
    average_speed(car,Design_Speed,Speed),
    speed_factor(car,Speed,SE),                    /* speed effects */
    grade_factor(car,Topo,GE),                     /* grade effects */
    fuzzy_multiply(SE,GE,COC).

truck_operating_cost(Design_Speed,Topo,COC) :-
    average_speed(truck,Design_Speed,Speed),
    speed_factor(truck,Speed,SE),                  /* speed effects */
    grade_factor(truck,Topo,GE),                   /* grade effects */
    fuzzy_multiply(SE,GE,COC).

yearly_time_cost(TPerc,CCT,ADT,Speed,Wage,Yearly_Time_Cost) :-
    fuzzy_add(TPerc,CCT,Comm_Trav_Perc),
    fuzzy_multiply(ADT,365,Yearly_Total),
    fuzzy_multiply(Yearly_Total,Comm_Trav_Perc,Yearly_Comm_Traffic),
    fuzzy_multiply(Yearly_Comm_Traffic,Wage,Over),
    fuzzy_divide(Over,Speed,YTCD),                         /* yearly cost ($) */
    fuzzy_divide(YTCD,1000,Yearly_Time_Cost), !. /* ($1,000) */

/*
these are used to help assess several types of impact
*/
average_speed(car,Design_Speed,S) :-
    !,
    fuzzy_multiply(Design_Speed,[0.9,1],S).
average_speed(truck,Design_Speed,S) :-
    !,
    fuzzy_multiply(Design_Speed,[0.45,0.55],S).

speed_factor(car,Speed,SF) :-
    !,
    fuzzy_divide(Speed,2857,SF1),
```

```
    fuzzy_add(SF1,[0.05,0.06],SF).
speed_factor(truck,[Speed,Sa,Sb],[0.49,0.53]) :-
    !,
    fuzzy_divide(Speed,476,SF1),
    fuzzy_add(SF1,[0.045,0.05],SF).

/* ----------------------------------------
    this determines the value of a given attribute based on
information contained on the PDF file.
*/
region_data(Point,Data) :-
    area(Area;Attributes),
    member(Data,Attributes),
    gc(full),
    in_area(Point,Area), !.
region_data(Point,Data) :-
    (
        default(Data);
        Data =.. [Functor,unknown]
    ), !.

/* ----------------------------------------
this predicate will scale (from 0 to 1) all elements of a given
impact list.
*/
scale_list([],[]) :-  !.
scale_list([Head|Tail],[Head1|Tail1]) :-
    Head =.. [Type,In],
    get_limits(In,Type,Best,Worst),
    scale_0_to_1(Best,Worst,In,Out),
    Head1 =.. [Type,Out],
    scale_list(Tail,Tail1).

get_limits(In,Type,Best,Max) :-          /* new 'worst' limit */
    limits(Type,Best,Worst),
    fuzzify(In,[Im,Ia,Ib]),
    Max is Im+Ib,
    Max > Worst, !,
    retract(limits(Type,Best,Worst)),
    assertz(limits(Type,Best,Max)), !.
get_limits(In,Type,Min,Worst) :-         /* new 'best' limit */
    limits(Type,Best,Worst),
    fuzzify(In,[Im,Ia,Ib]),
    Min is Im-Ia,
    Min < Best, !,
    retract(limits(Type,Best,Worst)),
    assertz(limits(Type,Min,Worst)), !.
```

```
get_limits(In,Type,Best,Worst) :-
    limits(Type,Best,Worst), !.

/*
Facts.Ari
==========================================
* This file contains default values that can be used directly by the
  evaluation predicates.
* The default values can be overriden if the user chooses to specify
  them in the PDF (Project Description File).
* The PDF file should be reconsulted AFTER this one is read in.
* Contains also the "obvious" clauses. If "obvious(Goal)." exists,
  then the "how" predicate will omit explanations about "Goal".
----------written in Nov.86 by Bernardo de Castilho
updated daily...
==========================================
*/

/* ----------
These are defaults. Should be used only when the user can't
supply accurate values:
*/
default(design_speed(100)).
default(number_of_lanes(4)).
default(trucks_with_hazardous_loads([0.05,0.08])).
default(truck_percentage([0.12,0.01,0.01])).
default(adt([10000,12000])).
default(density([5,6])).
default(interchange_cost([950,1000])).
default(underpass_cost([198,205])).
default(extra_cowboy_cost(100)).

/* ----------
These are global values. Can be overriden by user if more
accurate values -- fuzzy perhaps -- are available:
*/
interest_rate(0.1).
project_life(25).
driver_wage(12).
commercial_car_travel(0.05).

/* ----------
    Relative weights:
    These are arbitrary values. They represent the relative importance
of each attribute and are necessary to evaluate trade-offs between
different type of impact.
```

The values do not have to add up to any specific value, in fact they can be fuzzily specified also.

```
/* Environmental Attributes ----------*/
relative_weight(physical,1).
relative_weight(biotic,0.8).
relative_weight(aesthetic,0.8).
/* Social Attributes ----------*/
relative_weight(relocation,1).
/*Economic Attributes ----------*/
relative_weight(construction,[1,0,0.1]).
relative_weight(crossings,[1,0,0.1]).
relative_weight(operational,1).
relative_weight(growth,[0.5,0.6]).
relative_weight(land_value,0.5).

/* ----------------------------------------------
    These are normally used estimates: ($1000/km)
*/
land_cost(urban,[600,700]).
land_cost(cemetery,[480,560]),    *** etc ***
land_cost(unknown,[60,700]).
/* ignorance is paid for with uncertainty */
/* ----------------------------------------------*/
earthwork_cost(flat_no_bedrock,[0,60]).
earthwork_cost(flat_bedrock,[120,140]).    *** etc ***
earthwork_cost(unknown,[0,840]).
/* ---------------------------------------------- */
foundation_cost(drumlin,[312,364]).
foundation_cost(sandy_till,[318,371]).    *** etc ***
foundation_cost(unknown,[0,700]).

/* ----------------------------------------
these are used for vehicle operating costs
*/
grade_factor(car,flat_no_bedrock,1).
grade_factor(car,flat_bedrock,1).    *** etc ***
grade_factor(truck,unknown,[1,2]).

/* ----------------------------------
used for accident cost estimates:
(from the AASHO red book. these values represent km traveled per
accident dollar spent.)
*/
accident_cost_factor(Lanes,[880,1025]) :-   Lanes =< 2, !.
accident_cost_factor(_,[762,898]) :-   !.
```

```
/* ----------------------------------------
used for environmental impact assessment:
*/

waterbody_importance(water_supply,1).
waterbody_importance(wildlife_support,0.7).    *** etc ***
waterbody_importance(unknown,[0,1]).
/* ---------------------------------------- */
sensibility_to_pollution(urban,10).
sensibility_to_pollution(forest,5).    *** etc ***
sensibility_to_pollution(unknown,[2,5]).
/* ---------------------------------------- */
visual_context(Point,C) :-
     region_data(Point,land_use(U)),
     visual_c(U,C).
visual_context(_,C) :-
     visual_c(unknown,C).
visual_c(urban,0.16).
visual_c(cemetery,[0.12,0.15]).    *** etc ***
visual_c(unknown,[0.12,0.36]).
/* ---------------------------------------- */
species_importance(deer,1).
species_importance(geese,0.8).    *** etc ***
species_importance(unknown,[0.4,1]).

/* ----------------------------------------
used by explanation facilities to spare the user some obvious
conclusions and save time/stack/patience resources.
*/
obvious(in_area(_,_),
     $ Well, the point IS in the area, isn't it?$).
obvious(member(_,_),
     $ Oh God, are you an engineer?$).
obvious(append(_,_,_),
     $ Yes, I agree. This is really complex...$).
obvious(fuzzify(_,_),
     $ see Prade & Dubois, "Operations on fuzzy numbers".$).
obvious(fuzzy_add(_,_),
     $ see Prade & Dubois, "Operations on fuzzy numbers".$).
obvious(fuzzy_add(_,_,_),
     $ see Prade & Dubois, "Operations on fuzzy numbers".$).
obvious(fuzzy_subtract(_,_,_),
     $ see Prade & Dubois, "Operations on fuzzy numbers".$).
obvious(fuzzy_multiply(_,_,_),
     $ see Prade & Dubois, "Operations on fuzzy numbers".$).
obvious(fuzzy_divide(_,_,_),
     $ see Prade & Dubois, "Operations on fuzzy numbers".$).
```

```
obvious(fuzzy_power(_,_,_),
    $ see Prade & Dubois, "Operations on fuzzy numbers".$).
obvious(series_present_worth(_,_,_,_),
    $ This is fuzzified engineering economics.$).
obvious(scale_0_to_1(_,_,_,_),
    $ This is fuzzified arithmetics.$).

/* ----------------------------------------------
Limits: these values are used to scale the calculated attributes
Will be adjusted by the system if the evaluation predicated are
changed, but won't be automatically saved.
The format is 'limits(impact_type,Best,Worst_possible_value)'.
*/

/* Environmental Attributes ----------*/
limits(physical,0,8455).
limits(biotic,0,3.3).    *** etc ***
limits(land_value,0,1).
```

USER INTERFACE MODULE

```
/*
Hla.Ari
(Highway Location Assistant)
==========================================
User Interface -- Saves lots of typing too.
It reconsults all modules and handles the main computation blocks.
==========================================
 written in Nov.86 by Bernardo de Castilho
==========================================
*/

/*
This predicate might handle all the menu-driven user interface.
Alternatively, the user can exit to Prolog and query the system
directly.
*/

init :-
    cls,
    write($Consulting System Files... $),
    write($Inference, $),          [-infereng],
    write($Utilities, $),          [-utility],
    write($Knowledge  $),          [-kbase],
    nl,
    write($Consulting Data File... $),
    write($test.pdf$),             [-'test.pdf'], !.

point(Point) :-
    evaluate_point(Point,Impact_List,Aggregated_Impact),
    display_point_impacts(Point,Impact_List,Aggregated_Impact), !.

segment(P1,P2) :-
    /* find all point impacts----------*/
    get_central_point(P1,P2,CentralPoint),
    findall(I,evaluate_attribute(CentralPoint,I),RPIL),
    gc(full),
    /* find all linear entity impacts----------*/
    findall(Limp,evaluate_linear(P1,P2,Limp),RLIL),
    gc(full),
    /* put them all together----------*/
    merge_list(RPIL,RLIL,RIL),
    scale_list(RIL,SIL),
    gc(full),
    /* aggregate----------*/
    aggregate_impacts(SIL,LAG),
    distance(P1,P2,D),
```

```
    fuzzy__multiply(LAG,D,Total),
    gc(full),
    display__seg__impacts(P1,P2,SIL,Total), !.

get__central__point(p(X1,Y1),p(X2,Y2),p(X,Y)) :-
    X is abs((X2-X1)/2),
    Y is abs((Y2-Y1)/2), !.

display__point__impacts(Point,Impact__List,Aggregated__Impact) :-
    cls,
    write($
   +-------------------------------------------------------------------------+
   | *** Point Impact Assessment ***
|
   +-------------------------------------------------------------------------+
   | Point:
|
   +-------------+-----------------------------------------------------------+
   | Impact Type |0    0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8    0.9
1|
   +-------------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+$
    ),
    tmove(4,12), write(Point),
    tmove(7,80), nl,
    display__impact__list(Impact__List),
    display__total(Aggregated__Impact), !.

display__seg__impacts(P1,P2,Impact__List,Aggregated__Impact) :-
    cls,
    write($
   +-------------------------------------------------------------------------+
   | *** Highway Segment Impact Assessment ***
|
   +-------------------------------------------------------------------------+
   | Segment:
|
   +-------------+-----------------------------------------------------------+
   | Impact Type |0    0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8    0.9
1|
   +-------------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+$
    ),
    tmove(4, 14), print([P1,$ - $,P2]),
    tmove(7,80), nl,
    display__impact__list(Impact__List),
    display__total(Aggregated__Impact), !.

display__impact__list([]) :-  !.
```

```
display_impact_list([Head|Tail]) :-
    Head =.. [Functor,Value],
    /* heading */
    string_term(S,Functor),
    string_length(S,Len),
    Tab is 13-Len,
    print([$    | $,S,tab(Tab),$3$]),
    /* bars */
    fuzzy_multiply(Value,58,[Vm,Va,Vb]),
    Lo is integer(Vm-Va),
    Hi is integer(Vm+Vb),
    Av is integer(Vm),
    !,
    (
        for(X,0,58),
            writeone(X,Lo,Hi,Av),
        fail
    ;
        Hi>58, write($+$)
    ;
        write($|$)
    ),
    nl,
    display_impact_list(Tail).

writeone(X,_,_,X) :-   write($ $), !.
writeone(X,Lo,Hi,Av) :-
    (
        X < Lo
    ;
        X > Hi
    ),
    write($z$), !.
writeone(X,_,_,Av) :-
    X < Av,
    write($.$), !.
writeone(X,_,_,Av) :-
    X > Av,
    write($/$), !.

display_total(X) :-
    print([$    $,
  $+-------------+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
    | ** Total          |
|
    +-------------+-------------------------------------------------------+$
    ]),
```

```
    tget(L,_),
    dec(L,L1),
    tmove(L1,20),
    write(X),
    tmove(L,80), !.
```

SEARCH MODULE

```
/*
Module SEARCH
Search.Ari
==========================================
This is a Prolog implementation of a modified version of Dijkstra's
shortest path algorithm.
The links are dynamically generated, the search area and step
size are defined by the user.
==========================================
written in Nov.86 by Bernardo de Castilho
   (update   3.12: dynamic link generation)
   (update   4.12: ported from Turbo to Arity)
   (update  15.01: avoid redundant candidate evaluation)
==========================================
*/

/*----------------------------------------
This predicate checks the validity of the endpoints, forms the
search tree and displays the final route.
*/
search_path(Step,Origin,Destination) :-
    search_area(A),                                  /* make sure endpoints
*/
    in_area(Origin,A),                              /* are in search area   */
    in_area(Destination,A),
    retractall(link(_,_,_)),
    assertz(link(Destination,Destination,0)),    /* dummy link */
    retractall(step(_)),
    assert(step(Step)),
    retractall(tree_end(_)),
    assert(tree_end(Origin)),
    path(Destination,Origin),
    !,
    nl,
    write($*** Path located$), nl,
    link(_,Origin,I),
    print([$*** Cumulative Impact is $,I,nl]),
    go(Origin,Destination), !.                       /* display route */
search_path(_,O,_) :-
    search_area(A),
    not in_area(O,A),
```

```
    print([nl,$The specified origin ($,O,$) is not in the search area.$]),
    fail.
search_path(_,_,D) :-
    search_area(A),
    not in_area(D,A),
    print([nl,$The specified destination ($,D,$) is not in the search area.$]),
    fail.
search_path(_,_) :-  !.

/*
This predicate interprets the route.
*/
go(P,P) :-  print([nl,$*** That's it!$,nl]).
go(P1,P2) :-
    link(Next,P1,_),                    /* reverse! */
    print([$Going from $,P1,$ to $,Next,nl]),
    keyb(_,_),
    go(Next,P2).

/*
This one creates a tree of 'links(Origin,Dest,Cum_Impact)'.
*/
path(Origin,Destination) :-
    repeat,
    [!
        print([nl,$Expanding path...$,nl]),
        expand_path,                                    /* expand tree */
        gc(full),
        print([$Path Expanded...$,nl])
    !],
    link(_,Destination,_).                  /* repeat if not done */
path(_,_) :-  !.

expand_path :-                  /* find candidates and impacts */
    find_link(From,To,Impact),
        print([$Asserting $,From,$ -> $,To,$ ($,Impact,$)$,nl]),
        assertz(candidate(From,To,Impact)),
    fail.                       /* find all possible links */
expand_path :-                  /* change their status */
    find_lowest_impact(Impact),
    print([$Lowest impact is : $,Impact,nl]),
    candidate(From,To,Impact),
    assertz(link(From,To,Impact)),
    retract(candidate(From,To,Impact)),
    print([$*** New Link: $,From,$ -> $,To,$ ($,Impact,$)$,nl]),
    fail.                       /* repeat for all candidates */
expand_path :-  !.
```

```
find_lowest_impact(Impact) :-
    findall(I,candidate(_,_,I),L),
    sort(L,[Impact|Rest]).

find_link(From,To,Impact) :-
    link(_,From,_),
    find_dest(From,To),
    not link(_,To,_),
    not candidate(From,To,_),
    link(_,From,PCI),
    evaluate(From,To,LinkImpact),
    Impact is LinkImpact+PCI.

find_dest(p(X1,Y1),p(X2,Y2)) :-
    step(Step),
    tree_end(p(Xe,Ye)),
    distance(p(Xe,Ye),p(X1,Y1),D),
    (
        D =< Step,
        X2 is Xe,
        Y2 is Ye        /* end within reach, try it */
    ;
        X2 is X1+Step,
        Y2 is Y1
    ;
        X2 is X1-Step,
        Y2 is Y1
    ;
        Y2 is Y1+Step,
        X2 is X1
    ;
        Y2 is Y1-Step,
        X2 is X1
    ;
        X2 is X1+Step,
        Y2 is Y1+Step
    ;
        X2 is X1-Step,
        Y2 is Y1-Step
    ;
        Y2 is Y1+Step,
        X2 is X1-Step
    ;
        Y2 is Y1-Step,
        X2 is X1+Step
    ),
```

```
    search_area(A),
    in_area(p(X2,Y2),A).

evaluate(p(X1,Y1),p(X2,Y2),Z) :-
    XC is (X1+X2)/2,
    YC is (Y1+Y2)/2,
    distance(p(X1,Y1),p(X2,Y2),Length),
    /* find all point impacts ----------------------------*/
    findall(I,evaluate_attribute(p(XC,YC),I),RPIL),
    /* find all linear entity impacts ----------------------*/
    findall(Limp,evaluate_linear(p(X1,Y1),p(X2,Y2),Limp),RLIL),
    /* put them all together ------------------------------*/
    merge_list(RPIL,RLIL,RIL),
    scale_list(RIL,SIL),
    gc(full),
    /* aggregate ------------------------------------------*/
    aggregate_impacts(SIL,LAG),
    fuzzy_multiply(Length,LAG,[Z,_,_]),
    gc(full),
    print([$ *** Z: $,Z,nl]), !.
```