Queries and Their Application to Reasoning with Remote Sensing and GIS

David G. Goodenough, Daniel Charlebois[1], Stan Matwin[1], Doug MacDonald, and Alan J. Thomson
Natural Resources Canada
506 West Burnside Road
Victoria, B.C. V8Z 1M5, Canada
T: 604-363-0776 F: 604-363-0775 EMail: dgoodenough@a1.pfc.forestry.ca
[1]Department of Computer Science
University of Ottawa
Ottawa, Ontario K1N 6N5, Canada
T: 613-564-5069 F: 613-564-9486 Email: {stan, daniel}@csi.uottawa.ca

ABSTRACT

An intelligent system is being developed which integrates remote sensing data from aircraft and satellites with raster and vector geographic information systems (GIS). This System of Experts for Intelligent Data Management (SEIDAM) responds to queries or to product requests to select the appropriate mix of sensors, data processing methods and GIS to provide the answers. Recently, natural language processing was introduced into SEIDAM as one of the modes by which queries can be asked. The other two modes are selection of a stored query from a library or selection of subjects and attributes to create a query. Since SEIDAM contains more than one terrabyte of data, queries must also deal with meta data about the platforms, their sensors, and their properties. The long-term knowledge for SEIDAM is stored in frames which are object-oriented structures with multiple inheritance. Examples of queries are: "How much forest is there in this test site?' or "What are the wavelengths of the MODIS airborne simulator?" or "What remote sensing data are available for Clayoquot Sound before 1985?". A query is parsed in order to extract a set of goals that are passed to SEIDAM's reasoning system. Case-based reasoning and goal-regression are applied together to form a plan that, when executed, will satisfy the goals. A plan may involve the use of several expert systems that understand the use of GIS and the analysis of remotely sensed imagery. For the query "How much forest ...", for example, SEIDAM creates a plan that would check the forest inventory GIS and evaluate the timeliness of the corresponding GIS files (maps). Older GIS files would be updated using satellite data such as Thematic Mapper, prior to responding to the query. The presentation format of the answers must be matched with the characteristics of the questioner. SEIDAM also allows the user to specify one or more products to be created. These products likely meet the need of a collection of queries. Products supported include: updated GIS files, temporal change products, maps (forest cover, timber volume, canopy chemistry, infected trees, etc.), enhanced imagery, values and tabular summaries, text, processing and dissemination histories. This paper describes the flow of a query in SEIDAM, the structure for processing this query, and the application of case-based planning.

INTRODUCTION

In the latter part of this decade, users will be overwhelmed by the amounts of available satellite remote sensing data. SEIDAM (System of Experts for Intelligent Data Management) is conceived as a system to utilize only the data necessary to provide an answer to a user's question. SEIDAM also supports the users' demands for products, such as GIS file updates with remote sensing. An overview of query processing is presented as a guide to query format and type.

A query must pass through five processing stages before an answer is available (Thomson, Goodenough, 1994). These stages are: the query parser, processor selection, the processor, processor output, and the output translator to the answer. Basic queries are What? Where? When? Why? How? Who? In some cases the converse of Why?; i.e. Why not? is the focus of the question. We deal only with What? Where? When? How? and Who? types of questions, noting that "How much...?" and "How many...?" are "What?" types of questions rather than "How?" types of questions which require an explanation rather than a value. "Why?" will be included in terms of explanation by the expert systems, but not in terms of explaining policy decisions. Policy decisions belong to the user. Example questions considered in SEIDAM are as follows:

How much forest do (or did or will) we have?
What is the forest productivity expected on this site over the next 200 years?
Where has bud flush been reduced by damage agents?
What is the present timber volume of Douglas fir in this watershed?
What is the yearly rate of forest depletion in the Greater Victoria Watershed over the past 20 years?
Who was the source of the data used to provide this answer?
What would happen to the forest cover if the annual allowable cut was increased by 20%?
Which sensor is most suitable for measuring the amount of selective harvesting in Parson?
...

Interviews were held with domain experts in the B.C. Ministry of Forests and the B.C. Ministry of Environment, Lands and Parks. As a result, a long list of typical queries applicable to remote sensing and GIS were developed. These queries and the methods to obtain the answers represent a case. Ideally, the query parser would understand the English expression through a natural language translator system (NLS). Use of NLS with GIS is described by Krzanowski (1990), who covers the major issues in such systems, including ellipses, parsing time, and fuzzy requests. Ellipses are where a question or instruction to the system obtains its context from previous questions or instructions. Fuzzy requests are questions which include terms such as "near" or "in the vicinity of". A special type of ellipsis of major importance to simulation modelling is questions of the form "What would happen if ...". Krzanowski suggests that existing NLS are not powerful enough for use with spatial information systems. Note that SEIDAM adds a temporal dimension to the spatial dimensions.

The query parser must redefine the query into goals which the problem solver can understand in order to select the lower level expert systems and processors to answer the query. The parser should probably build on previous queries (ellipses), although this functionality has not yet been developed. The user can also command SEIDAM to create products which support a multitude of queries. SEIDAM also can use a prearranged set of queries which are stored in a query library. The capability to support the use of fuzzy queries, such as "What is the logging distribution near this park?" where "near" is a fuzzy term, will be added this year.

The issue of selecting a processor to answer a query is based on the fact that we have a choice of models, a choice of GIS platforms and a terrabyte of aircraft and satellite remote sensing data from multiple optical and radar sensors. There are a large number of processing methods from which to choose. We need to match parsed query elements with outputs possible from specific processes. Meta knowledge about processes and data must be available to the system. SEIDAM dynamically selects data sources (Fig. 1) (Thomson, 1994) in order to achieve the user goals defined by the initial queries and possible additional interactions with the user.

Expert system frames contain meta knowledge about the available processes and their required inputs and possible outputs. The outputs of some processes can be used as the inputs to others. A problem solver has been developed which can use this meta knowledge to generate the process call sequence for a particular output. When there is more than one method of obtaining the

0-7803-1497-2/94 $4.00 © 1994 IEEE

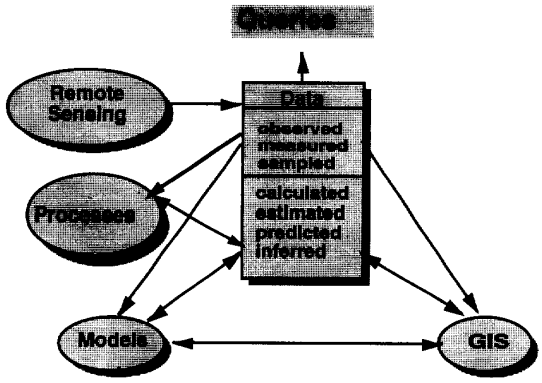

Figure 1. The complexity of using data to answer queries.

required output, the problem solver using case-based reasoning selects the appropriate alternative.

When the output from a series of processes has been obtained as a result of a query, it must be presented to the user in the most appropriate way. The output translator which puts the answer into a format that the user can easily assimilate. For example, the query "how much forest will we have in 50 years in the Tofino Creek watershed?" requires the selection of existing forest cover GIS files, updating these files with the latest remote sensing measurements of forest attributes, predicting growth and yield over the next 50 years, and generating the output (answer). The user may wish to visualize the forest cover in three dimensions in order to estimate possible changes in a scenic corridor. Or the user may want a GIS file containing forest species distribution with timber volume estimates. Or the user may simply want a number, such as the number of hectares by species. The tradeoffs for the output translator in answering a query involve three components: mode of information presentation, task environment of the decision, and individual characteristics of the decision maker (Wheeler and Sharda 1991). Wheeler and Sharda (1991) describe an expert system for selecting presentation mode. SEIDAM offers a wide-range of visualization options. In order for the user to not be overwhelmed with, for example, GIS detail, the system needs to be able to generalize the outputs to the scales desired by the user. This research issue will be addressed in phase 2 of this project. In the area of visualization, our experience in multimedia presentations is valuable for suggesting animation approaches to complex visualizations, while the production of text output geared to specific user concepts and languages has been illustrated by Thomson and Taylor (1990).

QUERY CREATION

SEIDAM offers the user two ways to specify a goal: product selection from a list of available products and applications; and queries. Queries can be entered into SEIDAM using a natural language system. These queries can be interrogative, as given above, or imperative, such as "Describe the sensor characteristics of the LANDSAT 5 sensors." The NLS can interpret these queries into parsed goals on which the problem solver can operate. However, natural language processing has many pitfalls and this method of query input can easily breakdown if words (and concepts) are used which are not in the language dictionary.

A second way of specifying a query is through a windows-based set of choices. One window provides a list of often desired attributes, such as forest cover or timber volume. A second window allows a choice of modifiers such as average or maximum, etc. The user also specifies the time window of interest and the spatial location by symbolic name, map sheet number(s), or graphical area selection. The result is a query in English which represents the user goals.

A third way of selecting a query is to choose a query from the query library window. The user can alter this query to represent the current goals. For example, the query "What is the yearly rate of forest depletion in the Greater Victoria Watershed over the past 20 years?" could be modified to "What is the rate of forest depletion in Tofino Creek over the past 40 years?". Case-based reasoning would then use the case pertaining to the first query and modify it to create the operator (processor) sequence to achieve the desired answer. To achieve success in this system, we make use of planning and the merging of plans.

PLANNING (PROBLEM SOLVING) AND PLAN MERGING

The user specifies a conjunctive goal through a query. The planner starts with this conjunctive goal, as shown in figure 2. The approach taken is to solve each conjunct separately and then combine the solutions. Goal conjunctions can represent two types of goals: user goals and operator goals. User goal conjunctions are the initial goals that a problem solver must attempt to satisfy. Operator goal conjunctions, or subgoals, are the pre-conditions of operators that the problem solver is attempting to apply. A potential issue is that user goal conjunctions can introduce problems such as pre-condition clobbering (Sussman, 1973), whereas operator goal conjunctions should not. Initially, we assume that the operator goal conjunctions are correct. This assumption will be relaxed later. If a solution is found, we assume that it is correct, since it is based upon the domain knowledge entered into SEIDAM.

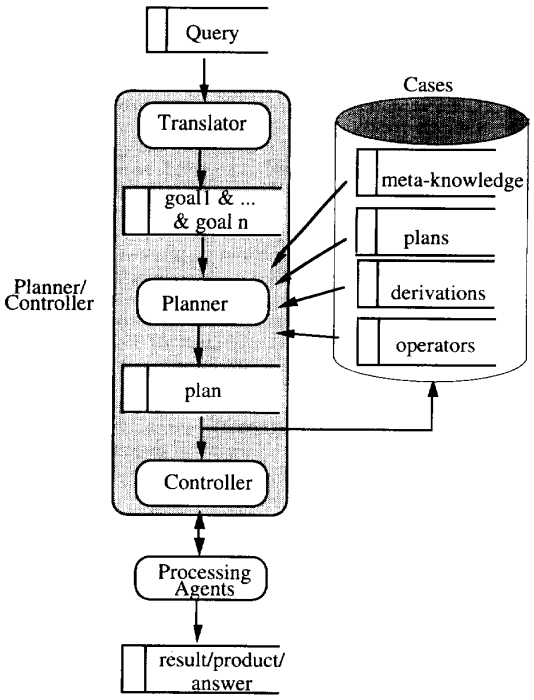

Figure 2. Architecture of the Problem Solving Environment.

Each goal in a conjunction is either independent of the others, dependent on the others, or shares resources with the others. Given a correct knowledge base, it should be possible to solve these types of goal conjunctions in the following ways:

1- independent goals: solve one goal from the initial state, then solve the second goal from the final state produced by the application of the solution to the first goal. Since goals are independent and operators prevent one solution from clobbering another, this approach will always produce a solution.

2- dependent goals: the order in which we attempt to satisfy each goal in this type of conjunction becomes important. Often, simply imposing an ordering on the goals might allow the problem solver to find a solution.

3- independent goals that share resources: solve first goal from the initial state, solve the second goal from the original initial state, and merge both solutions.

In accordance with the operator correctness assumption above, the problem solver should solve each user goal independently and then merge the solutions. Although this differs from the appending in 1 above, independent goals can be handled in this way since they do not share any resources. Operators from each solution can be applied while only considering the partial ordering in each solution. In 3, this is how we suggest goals should be handled. User goals that are not independent, can also be handled in this way since, as a result of merging, there is a partial ordering that is imposed on the operators. This leads to the application of a generic "solve algorithm".

The solve algorithm for multiple goals:

```
solve( ( Goal and Goals ), Solution ) if
   solve_one_goal( Goal, Sol1 ) and
   solve( Goals, Sol2 ) and
   merge( Sol1, Sol2, Solution ).
```

The solve_one_goal procedure uses one of three problem solving paradigms: transformational analogy, derivational analogy, and goal regression (Charlebois, 1993). To avoid searching a large state space, the procedure will first attempt to find a case based on the transformational analogy paradigm, that can completely satisfy the goal by applying modification rules to account for differences between the original case and the current goal. If a transformational case exists, then the system will solve the other subgoals and subsequently merge the solutions. If the system cannot find a transformational case, it must resort to knowledge-based searching. Hence, the system attempts to solve the goal by retrieving a derivational case. The purpose of this approach is twofold. First, where several solutions exist, a derivational case will provide the system the justification for selecting one over another. Second, since the search mechanism used here is goal-regression, derivational analogy allows the system to select operators that would otherwise be overlooked, thus expanding the scope of the search by considering yet more solutions. Finally, if there does not exist a case, either transformational or derivational, the system will use goal-regression.

Let us examine relaxing the operator correctness assumption by relaxing the order of the preconditions of the operator. If we disregard the assumption, the consequence is that satisfying the operator pre-conditions may not be possible. The ordering of an operator's pre-conditions also represents the ordering imposed on the operators used to satisfy them. If an operator is incorrect in the ordering of its pre-conditions, it cannot be used and possible solutions will not be considered. In SEIDAM use is made of AVS for visualization and PCI image analysis software. Consider the following operators for loading the results of an analysis with PCI into an AVS network. These operators are STRIPS-like operators with preconditions, add and delete lists ((Fikes, 1981), (Charlebois, 1993)).

```
avs_pci_set_red( Channel, Reply ) ::
   if          synched( avs ) and
               current_image( Filename )
   then delete synched( avs ) and
               all( red_channel( _ ) )
        add    red_channel( Channel ).

avs_pci_read_image( Filename, Reply ) ::
   if          loaded( image_display_net ) and
               synched( avs )
   then delete synched( avs ) and
               all( current_image( _ ) )
        add    current_image( Filename ).
```

There is a need to connect PCI image channels to AVS color channels. If an expert system submits the goal red_channel(1), goal-regression would first select the operator avs_pci_set_red, whose pre-conditions are synched(avs) and current_image(Filename). Goal regression would attempt to find a solution to synched(avs) first and then try to find a solution to current_image(Filename) afterward. Since the avs_pci_read_image operator deletes the fact synched(avs), it cannot be used. However, if the pre-conditions of the avs_pci_set_red operator were reversed, goal regression would succeed. By solving each conjunct independently and then ordering the operators by merging them, it is possible to find a solution to the complete conjunction even though the initial ordering would fail. The domain knowledge which led to the creation of these operators still has to be correct.

The plan merge algorithm applies techniques similar to those from the NOAH (Sacerdoti, 1975) algorithm as well as the algorithm introduced by Yang (Yang, 1992). The main objective is to combine two plans into one by way of a "sorting by merging" approach. The algorithm assumes that the operators in each plan are processed with respect to their respective partial ordering. The operators from both plans are examined to determine if additional ordering constraints are necessary when they are combined into one. This can be achieved by using a conflict critic as proposed by Sacerdoti and applying the interaction definitions proposed by Yang. The important difference between these authors and this works is that when a mutual clobbering situation arises, rather than failing, the merge algorithm will use two different methods to resolve the conflict: (1) select one of the plans and determine if, within the next k operators, the protected pre-conditions subject to the clobbering become unprotected; if this occurs then the merge can proceed; (2) if (1) fails, then check the case-base for a solution.

Mutual pre-condition clobbering occurs when two competing operators from different plans must be applied. As a result of applying one of the operators, some facts in the world are no longer true and thus prevent the application of the other operator. It is conceivable, that after applying several operators from one plan, that it will be possible to apply the operators from the other. In the merge algorithm (Analogical Plan Merge: APM(k)), a constant k represents the number of operators the system will examine before declaring a failure to merge two plans (Charlebois, 1993). If a solution cannot be found after examining k operators, the system will try to find a case that addresses the merge problem.

QUERY ANSWERING - AN EXAMPLE

In this section, we will step through a detailed example to illustrate the behavior of the system. The query presented to the system has two subgoals. The solution for the first subgoal is explained in sub-section below and shows how goal-regression and transformational analogy can be successfully integrated. In the second sub-section, we show how two solutions, one for each subgoal, can be merged into one solution although some operators may conflict. The system will be presented with the following query: "Create a map that shows the forest depletion in Clayoquot Sound over the past 20 years and create a map separating western cedar and Douglas fir canopy over Clayoquot Sound."

SEIDAM will provide a map (GIS files) of the desired site that contains the forest distribution per age and species dated 20 years ago, thematic-mapper (TM) and color infra-red geocoded imagery over the site, as well as image processing and visualization tools. To create the forest depletion map, the TM imagery would have to be processed to detect the areas representing depleted forest cover. Color infra-red imagery can be used to improve the boundaries discriminating between Douglas fir and western cedar.

Solving the First Subgoal

The system will first try to find a transformational analogy case that satisfies each subgoal. If it cannot find a transformational analogy case, it will try to find a derivational analogy case. If it cannot find a derivational analogy case, it will try goal-regression. For the example query above, we will show how the system produces a plan for the first part of the query conjunction; i.e. computing the forest depletion. If the system does not have a case

that can satisfy the query directly, but does have a case that classifies imagery and a case that can display a map, then it could use the operator given below by using goal-regression:

```
depletion_overlay( Site, Time ) ::
   if          image_classified( Site, Time ) and
               displayed_map( Site, Time )
   then add    depletion( Site, Time ).
```

The original goal is now replaced by this operator's pre-conditions and the system attempts to solve them separately. Since cases exist to solve each of the subgoals, they are retrieved, adapted to the current problem and subsequently merged into one solution.

If the solutions for the two pre-conditions for the depletion_overlay operator are:

```
subgoal: image_classified( Site, Time )

plan (1): start( visualization_program )
          load( tm_image( Site, Time ) )
          classify(tm_image(Site, Time))
          set_overlay( red, old_growth )
          set_overlay( green, depleted_area )

subgoal: displayed_map( Site, Time )

plan (2): start( visualization_program )
          load( map( Site, Time ) )
          set_level( forest_level )
          display( map( Site, Time ) )
```

then the merge algorithm would behave as follows:

Set the current operator from either plan to the first one and the time index to zero (the time index indicates the point of insertion of an operator into the final solution). The next step is to determine if the current operators are applicable at the current time index. In the example, both operators are applicable. The system then checks for operator redundancy; i.e. are they identical operators or is the subgoal for which they are used already satisfied by the side effects of previous operators in the final solution. In the example, the current operators, start(visualization_program), from both plans are identical resulting in the removal of one of them. If we assume that the deleted operator is from plan 2, then the current operator in plan 1 is still start(visualization_program) , the current operator from plan 2 is load(map(Site, Time)), and the final solution is empty. Since the current operator in plan 2 is not applicable, the current operator from plan 1 is and it does not clobber the pre-conditions of plan 2's current operator, the operator from plan 1 is inserted into the final solution. The current operator from plan 1 is now set to load(tm_image(Site, Time)). For the remaining operators from either solution, their are no conflicts thus resulting in:

```
start( visualization_program )                    plan A
load( map( Site, Time ) )
set_level( forest_level )
display( map( Site, Time ) )
load( tm_image( Site, Time ) )
classify(tm_image(Site, Time))
set_overlay( red, old_growth )
set_overlay( green, depleted_area )
depletion_overlay( Site, Time )
save_map( forest_depletion_cover_level )
```

At this point the depletion_overlay operator was appended. The most important consideration whenever an operator is inserted into the final solution is to update the current state description. All search based methods actually simulate the application of each operator. If a new one is added, the state description at the point of insertion, as well as at all the following time indices, must reflect the effect of the insertion.

Satisfying Both Subgoals

The system now has the following case to solve; namely to create a map that separates Douglas fir and western cedar forest covers:

```
start( visualization_program )                    plan B
load( map( Site, Time ) )
set_level( forest_level )
display( map( Site, Time ) )
load( color_ir_image( Site, Time ) )
segment(color_ir_image( Site, Time ) )
label_segments(color_ir_image( Site, Time ) )
color_ir_overlay( Site, Time )
save_map( canopy_cover_level )
```

For the sake of discussion, we assume only one image can be overlaid on a map at any given time. The system must now merge the plans, A and B, for the two subgoals in our example query. The behavior is similar to what is described above up until the system must load the imagery. At that point, there is mutual pre-condition clobbering since only one image can be overlaid at a time. In the previous section, we introduced the APM(k) algorithm. For this example, k is set to 6 and the merge proceeds as follows. The system would select one of the two solutions and append up to k of its operators to the final solution while checking at each step if the pre-conditions of the current operator of the other plan are still not satisfied. Suppose that the system chose the solution for the forest depletion first, plan A. The final solution this far would include every operator up to the conflicting ones and is shown in (1) below. In (2) below, by selecting plan B first, the system would start appending the operators to the final solution. If by the time the number of appended operators reaches k, the current operator in plan A is not applicable, then the system fails. However, while appending the operators the system discovered that the label_segments operator could not be applied because its preconditions were not satisfied; namely, the GIS did not contain labels, and so TM classification needed to be used to provide the labels for the high resolution, aerial photography segmentation. The system then backtracks to the point of conflict and tries to append plan A to the final solution, (3) below.

```
start( visualization_program )                    (1)
load( map( Site, Time ) )
set_level( forest_level )
display( map( Site, Time ) )          Conflict found with multiple
                                      image loads

start( visualization_program )                    (2)
load( map( Site, Time ) )
set_level( forest_level )
display( map( Site, Time ) )
load( color_ir_image( Site, Time ) )
segment(color_ir_image( Site, Time ) )   Conflict found with
                                         label segments since
                                         the GIS did not
                                         contain class labels
                                         which are supplied by
                                         TM classification.

start( visualization_program )                    (3)
load( map( Site, Time ) )
set_level( forest_level )
display( map( Site, Time ) )
load( tm_image( Site, Time ) )
classify(tm_image(Site, Time))
set_overlay( red, old_growth )
set_overlay( green, depleted_area )
depletion_overlay( Site, Time )
save_map( forest_depletion_cover_level )  Can now append rest
                                          of Plan B.
```

After appending the operators from plan A, k has been reached. However, since the pre-conditions of the current operator from plan B are now satisfied, the rest of that plan can now be appended to the final solution:

```
start( visualization_program )
load( map( Site, Time ) )
set_level( forest_level )
display( map( Site, Time ) )
load( tm_image( Site, Time ) )
classify(tm_image(Site, Time))
```

set_overlay(red, old_growth)
set_overlay(green, depleted_area)
depletion_overlay(Site, Time)
save_map(forest_depletion_cover_level)
load(color_ir_image(Site, Time))
segment(color_ir_image(Site, Time))
label_segments(color_ir_image(Site, Time))
color_ir_overlay(Site, Time)
save_map(canopy_cover_level)

Since Plan A has now performed the TM classification, the label_segments operator for the color imagery will now succeed. At this point we have expended no major processing costs in creating this plan. Moreover, a domain expert training the system could ask for another plan or modify this plan. This shows the flexibility for the system to construct a plan for a conjunctive query which integrates remote sensing and GIS data. This example plan could be applied and may fail if the accuracies of the desired result are not met. In this case, the problem solver would attempt to create a new plan if the meta knowledge about cases indicated that there were more successful, but more costly, approaches.

CONCLUSIONS

In this paper we began by describing query entry in the SEIDAM system. There are three means of entering queries: selection from a query library, natural language processing, and selection of query attributes. To obtain an answer to a query, a problem solver needed to construct a plan of analysis from a set of processes and data. This plan consisted of an ordered sequence of operators. This was followed by a description of the merging of plans which had operators with preconditions which were clobbered by the addition of other operators. An example demonstrating flexible merging of plans to respond to queries has been given. The problem solver is written in Prolog and has been demonstrated for some queries for Thematic Mapper imagery and GIS. Now under development is the addition of the aircraft sensor analysis and the growth and yield models.

ACKNOWLEDGEMENTS

The SEIDAM (System of Experts for Intelligent Data Management) Project is a project under NASA's Applied Information Systems Research Program (Joe Bredekamp, Glenn Mucklow) and we thank NASA for their on-going support and participation. The project is also supported by the Government of Canada (Natural Resources Canada, Industry Canada), the province of British Columbia (Ministry of Forests; Ministry of Environment, Lands and Parks), the EEC's Joint Research Centre at Ispra, and the Royal Institute of Technology in Stockholm, Sweden. We are most grateful for the contributions of these organizations and their co-investigators. Additional information on SEIDAM can be obtained by writing the authors or by accessing our World Wide Web site (http://pine.pfc.forestry.ca). We also wish to thank NSERC for its support for the research by Professor Stan Matwin and Daniel Charlebois.

REFERENCES

Charlebois, D., Goodenough, D.G., Matwin, S., "Machine Learning from Remote Sensing Analysis," in Proceedings of IGARSS-93 Symposium, 1993, pp. 165-172.

Fikes R., Hart P., Nilsson N., "Learning and Executing Generalized Robot Plans", in Readings in AI Ed. B.Webber and N.Nilsson, Palo Alto, CA, 1981.

Krzanowski, R.M. "Natural language interface to geographic databases. Experiments with Intelligent Assistant." in Proceedings of GIS '90 Symposium, Vancouver, B.C. March 1990, pp. 513-518.

Sacerdoti, E.D., "The Non-Linear Nature of Plans," In Proceedings of IJCAI-75, 1975, pp. 206-213.

Sussman, G. J., "A Computational Model of Skill Acquisition," Ph.D. Thesis, M.I.T, Cambridge, MA., 1973

Thomson, A.J. "Modelling considerations in the SEIDAM project." SEIDAM Workshop Report. 1994.

Thomson, A.J., Goodenough, D.G., "SEIDAM Queries and their Processing." SEIDAM Workshop Report. 1994.

Thomson, A.J., Taylor, C.M.A. . "An expert system for diagnosis and treatment of nutrient deficiencies of Sitka spruce in Great Britain." in AI Applications vol. 4, 1990, pp. 44-52.

Wheeler, B.C., and Sharda, R. "INFORMEX: An expert system to enhance information presentation." In: Knowledge-Based Systems and Neural Networks, Elsevier Science Publishing Co.R. Sharda et. al. (eds.), 1991. p. 99-115.

Yang, Q. "Merging Separately Generated Plans with Restricted Interactions," in Computational Intelligence, Volume 8, Number 4, 1992, pp. 648-676.