REASONING FOR AUTOMATED MODEL INTEGRATION

TING-PENG LIANG
Krannert Graduate School of Management
Purdue University, West Lafayette, Indiana 47907

This paper investigates some reasoning issues involved in developing an integrated modeling environment called a model management system. A model management system is a computer-based environment designed to support effective development and utilization of quantitative decision models. Since the construction of decision models is a knowledge-intensive process, reasoning plays a critical role. Reasoning is particularly important when automated model integration is needed in a large-scale system. In this case, heuristics are required to reduce the complexity of the process. This paper examines the planning and automated formulation of complex models from smaller building blocks. First, a hierarchy of abstractions that integrates previous contributions in model management is presented. Then, a modeling process is formulated as a planning process by which a set of operators are scheduled to achieve a specific goal. This process involves searches for alternatives that can eliminate the difference between the initial state and the goal state. Various reasoning strategies and heuristic evaluation functions that can be used to improve the efficiency of developing a master plan for model integration are discussed.

INTRODUCTION

A model management system (MMS) is a software system that handles all accesses to decision models stored in a model base. Its primary purpose is to enhance decision performance by facilitating the construction and utilization of decision models. The idea of model management is similar to that of data management except that the objects to be handled are decision models rather than numerical data. By "decision models" we mean quantitative models for problem solving, such as production schedule, inventory control, and facility management models. In most decision processes, the decision maker needs accurate data and reliable quantitative models. For example, a firm needs a data base to store all sales data and good sales forecasting and inventory control models to predict future sales and to reduce inventory costs. To achieve this goal, functions that support model creation, storage, retrieval, execution, and explanation are essential. Among them, model creation is the most knowledge-intensive function and requires knowledge of both individual models and the modeling process.

There are two different ways that a new model can be created: It can be developed from scratch, or it can be created by properly integrating existing modules. The most straightforward approach to solving a new problem often is

Applied Artificial Intelligence: 4:337–358, 1990

to develop a new model for the problem. This provides the decision maker with models tailored to the needs of the problem. In some situations, however, creating new models from scratch may not be necessary or feasible. For example, the problem encountered may be a combination of several smaller problems, each of which has an existing model. In this case, a proper integration of those existing models may be adequate for solving the problem. The process by which smaller models are integrated to form a composite model is called *model integration* (d'Alessandro et al., 1989; Dempster and Ireland, 1989; Geoffrion, 1989; Liang, 1988). It allows effective use of the stored models, saves modeling costs, and expedites the modeling process.

When a composite model is desired, the typical process involves at least three major stages: planning, evaluation, and selection. At the planning stage, proper models are identified and scheduled. At the evaluation stage, selected candidate models are evaluated and unqualified ones are deleted. At the selection stage, criteria are used to choose the most appropriate model if more than one candidate is available.

There are two approaches to the model integration process: user-directed modeling and automatic modeling. Both are supported to a certain extent by MMS. User-directed modeling allows a decision maker to specify how several smaller models can be combined to become a larger one. The user is responsible for finding a set of appropriate models and for determining the best way to integrate them. In other words, the user takes care of all three stages in the process, and the MMS serves only as a blackboard on which the user examines different alternatives. Because the system performs a limited number of intellectual activities in this case, a good model manipulation language may be adequate.

Automatic modeling, on the other hand, requires that the system automatically formulate a decision model for the user. A long-term goal of automatic modeling is to develop an MMS capable of designing decision models based on the problem description provided by the decision maker. This would allow the system to replace human model builders. Given current information technology, however, this goal is, at least for the present, very difficult to achieve. One major difficulty is that a modeling process is usually highly unstructured and involves a great deal of common sense, a set of knowledge computers cannot yet handle.

At present, a feasible goal for model management would be semi-automatic modeling. In other words, given the desired information, the system is capable of finding a model or a sequence of models already stored in the model base and of suggesting candidate models (planning). The user is responsible for selecting and validating the models formulated automatically by the system. After a formulated model is chosen by the user, the system executes the model and reports the result. Figure 1 illustrates this cooperation between the system and the user.

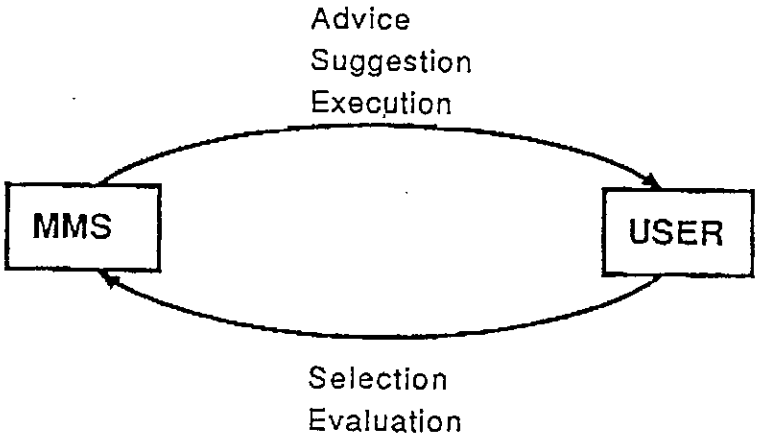

FIGURE 1. Cooperation between MMS and the user.

The primary purpose of this paper is to explore the reasoning issues involved in automated formulation of quantitative models. Developing the semi-automatic model integration capability needs both model representation schemes that logically represent each model in the model base and reasoning mechanisms that schedule models. In recent years, several model representation schemes have been developed, such as relational (Blanning, 1982, 1985a, b, 1986), logic-based (Bonczek et al., 1981; Dutta and Basu, 1984; Kimbrough, 1986; Krishnan, 1990; Pan et al., 1986), frame-based (Dolk and Konsynski, 1984), and graph-based approaches (Elam et al., 1980; Geoffrion, 1985, 1987; Liang, 1986). In this article, reasoning issues involved in the model integration process are discussed from a planning perspective.

Most problem-solving processes can be considered as planning processes through which a set of operators can be found and scheduled to eliminate the difference between the initial state and the goal state (Simon, 1981, 1983). This process involves a search of subgoals, operators, macro-operators, and abstractions. By an analogous definition, the model integration process is considered a process by which available models are selected and scheduled to eliminate the difference between the desired information and the available information. To determine and eliminate the difference, the following issues are crucial:

1. State representation: What information should be included in a state representation?
2. Reasoning: What mechanisms can be used to eliminate the difference between the goal state and the initial state?
3. Heuristic functions: How can the efficiency of the process be improved?

They will be discussed sequentially in the remainder of this article.

HIERARCHY OF ABSTRACTIONS: A REVIEW OF MODEL REPRESENTATION

To determine how a state should be represented, we must first consider what level of abstraction is necessary. Here the concept of abstraction is very important. To solve a complex problem efficiently, a problem solver must first ignore low level details and concentrate on the essential features of the problem. The details are filled in after the problem has been solved at a higher level. Therefore, abstraction formation involves loss of content, which makes the abstraction simpler than its instantiation(s) (Darden, 1987; Korf, 1987). This idea has been used in problem solving for a long time (Polya, 1957) and adopted by several general-purpose problem-solving programs, including General Problem Solver (GPS) (Newell and Simon, 1972) and ABSTRIP (Sacerdoti, 1974).

In the model management arena, Dolk and Konsynski (1984) first adopted the term "abstraction" and presented a model abstraction technique. In fact, despite grounding in different formalisms, most model representation schemes presented in previous research reflect different levels of model abstraction, from user-oriented to execution-oriented. For example, Blanning (1982, 1985a, b, 1986) emphasizes the manipulation of data relations (data level); Liang (1985) focuses on the mapping between inputs and outputs (model level); Geoffrion (1985, 1987) presents a hierarchical framework for structured modeling (structure level); Dolk (1986) and Dolk and Konsynski (1984) concentrate on model specification (specification level). Figure 2 illustrates those five levels of abstraction for the EOQ model that calculates economic order quantity from demand, holding cost, and ordering cost.

Program Level

At the bottom of the hierarchy is a Pascal implementation of the economic order quantity (EOQ) model. This reflects a highly machine-oriented view of decision models. The advantage of this representation is that the model can be stored in a model base and be readily integrated and compiled for execution. Its major disadvantage, however, is that little information relevant to model management is provided. For example, it provides little input and output information and can be activated by the model name only.

Specification Level

By ignoring some implementation details, Dolk and Konsynski (1984) developed a frame-based model abstraction technique. This technique represents models by their data objects, procedures, and assertions, all expressed in first-

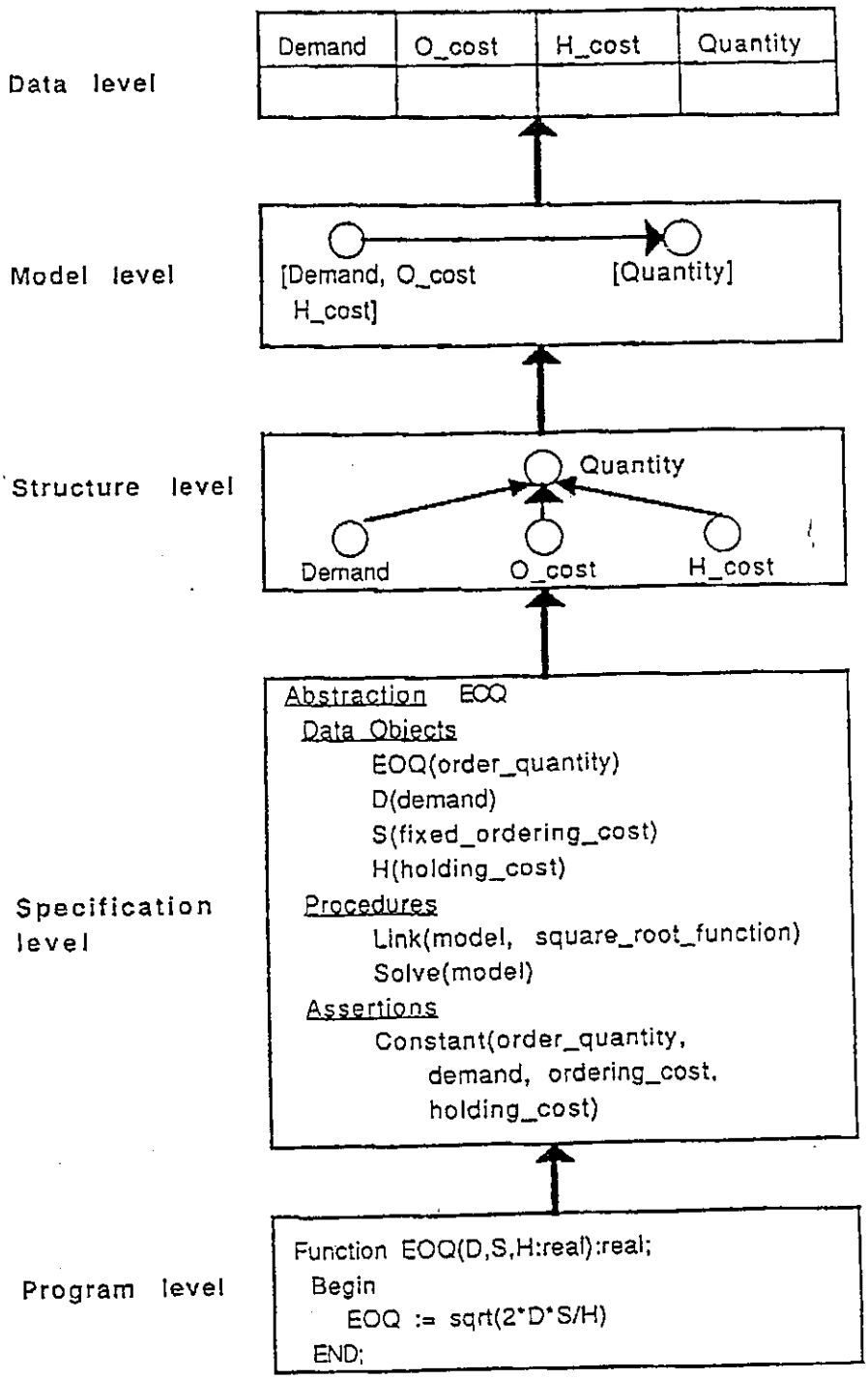

FIGURE 2. Hierarchy of model abstraction.

order predicate logic. An abstraction for the EOQ model is shown at the specification level in Fig. 2.

The contents lost at this level of abstraction are implementation details and direct computer executability. However, it gains computer language independence. The same specification may have interfaces to programs in different computer languages. An early work conducted by Elam et al. (1980) also represented models at a similar level of abstraction but in a graphical form. They adopted the concept of SI nets in artificial intelligence.

This level of model abstraction provides information useful to model builders. From a modeling perspective, however, this information can be further stratified based on its relative priority. For example, some low-level operations, such as checking data formats, may substantially increase the complexity of model integration and need not be considered until a set of potential model combinations has been identified. Therefore, further abstraction is desirable.

Structure Level

Geoffrion's framework for structured modeling further eliminates some integrity constraints and data formats. It portrays the fundamental structure of a model by its elemental structure, generic graphs, and modular structure.

One feature of structured modeling is the use of graphical symbols rather than text-based specifications, which make the functional relationships among various modules very clear. In Geoffrion's framework, nodes stand for modeling elements and arcs stand for calls. A modular structure of the EOQ model, as shown in Fig. 2, contains four nodes and three arcs.

A graph-based model structure provides certain insights into a model. For the purpose of model integration, however, representing the detailed structure may not be necessary in many situations. This is particularly true when the model is nondecomposable. For example, the mapping between demand and order quantity in the EOQ model cannot be used independently unless the ordering cost and holding cost are also present. In other words, those three data attributes, in combination, determine the EOQ. Therefore, the three arcs of the EOQ model are highly dependent and can be combined.

Model Level

By considering each model stored in the model base as a bridge or a mapping between input and output, Liang's approach uses a node to represent a set of data attributes and an arc to represent a set of functions that can be used to convert from one node to another. Since a model is composed of inputs, outputs, and a set of functions for converting inputs to outputs, it can be represented as a

combination of two nodes and one arc. For example, the EOQ is a mapping between two sets, [Demand, O_cost, H_cost] and [Quantity].

Because each model in the model base is considered a single mapping, this approach allows an associated cost or validity value to be estimated for each model. This is important when algorithms and heuristics in graph theory are used for model integration (Liang and Jones, 1988).

Data Level

At the top of the hierarchy is a relational framework of models. Instead of adopting artificial intelligence techniques, Blanning (1982, 1985a, b, 1986) focuses on model manipulations including join and projection. It reflects a user-oriented view of model management: The user obtains the desired information without all the details of calculation. For example, the EOQ model is considered a relation composed of demand, holding cost, ordering cost, and quantity at this level.

Model Integration Process

The hierarchical view of model abstraction indicates that an automated model integration process should include the following steps:

1. Identify the desired information.
2. Develop a master plan for building a composite model when there is no single model available for producing the desired information. The master plan determines the sequence by which a set of models should be executed.
3. Determine the model structure and retrieve the corresponding model specifications according to the master plan.
4. Evaluate the feasibility of the master plan by checking details such as data format and model assumptions.
5. Combine selected programs to formulate an executable model if the master plan is proven feasible. Otherwise, go to step 2 to develop another plan.

For example, if the inventory holding cost is not a constant but determined by a holding cost model with interest expenses and warehouse operation costs as its inputs, then calculating the economic order quantity involves an integration of two models: EOQ and the holding cost model. First, the MMS finds that the table (shown at the top of Fig. 3) requested by the user contains information from more than one model. Then the system develops a master plan for executing the selected models. In this example, the sequence is (1) the holding cost model and (2) the EOQ model. On the basis of master plan the structure of the composite model can be determined and validated. By "validated" we mean that

all integrity constraints and data formats are checked to assure that the developed model is suitable for a particular decision. Figure 3 shows the corresponding representations at the data level, model level, and structure level. Finally, specifications and executable programs can be activated and executed.

REASONING FOR MODEL INTEGRATION

The key step in the model integration process described above is the development of a master plan. Unless a sequence of models is found capable of producing the desired information, there is no need to check the integrity constraint or data format. Therefore, each state in the modeling process can be represented as a set of data attributes. Each state in the modeling process can be represented as a set of data attributes. Each model stored in the model base is considered an operator that converts one state into another. For instance, the EOQ model is an operator that converts the state [Demand, O_cost, H_cost] to another state [Quantity].

By those definitions, the initial state is composed of all available information and the goal state is the information desired by the decision maker. The development of a master plan for model integration is a process in which operators are scheduled to convert the initial state to a specified goal state by sequentially achieving a set of subgoals.

Two issues are crucial to developing such a master plan. First, how can the proper candidate models be selected from the model base? Second, how can the selected models be scheduled efficiently?

Selection of Candidate Models

Model selection is a two-stage process. First, given the desired information, existing models that may be useful for constructing the composite model must be identified. This process includes the following procedures:

1. Determine whether a model can produce all or part of the desired information.
2. Check the availability of the model in the model base.
3. Compare the identified models to eliminate dominated models. This step reduces the number of candidate models and simplifies the process of model scheduling.
4. Check model assumptions and other integrity constraints based on the problem description to reduce the number of candidates further.

After locating candidate models, the MMS schedules these primitive models to formulate a composite model. When more than one composite model is avail-

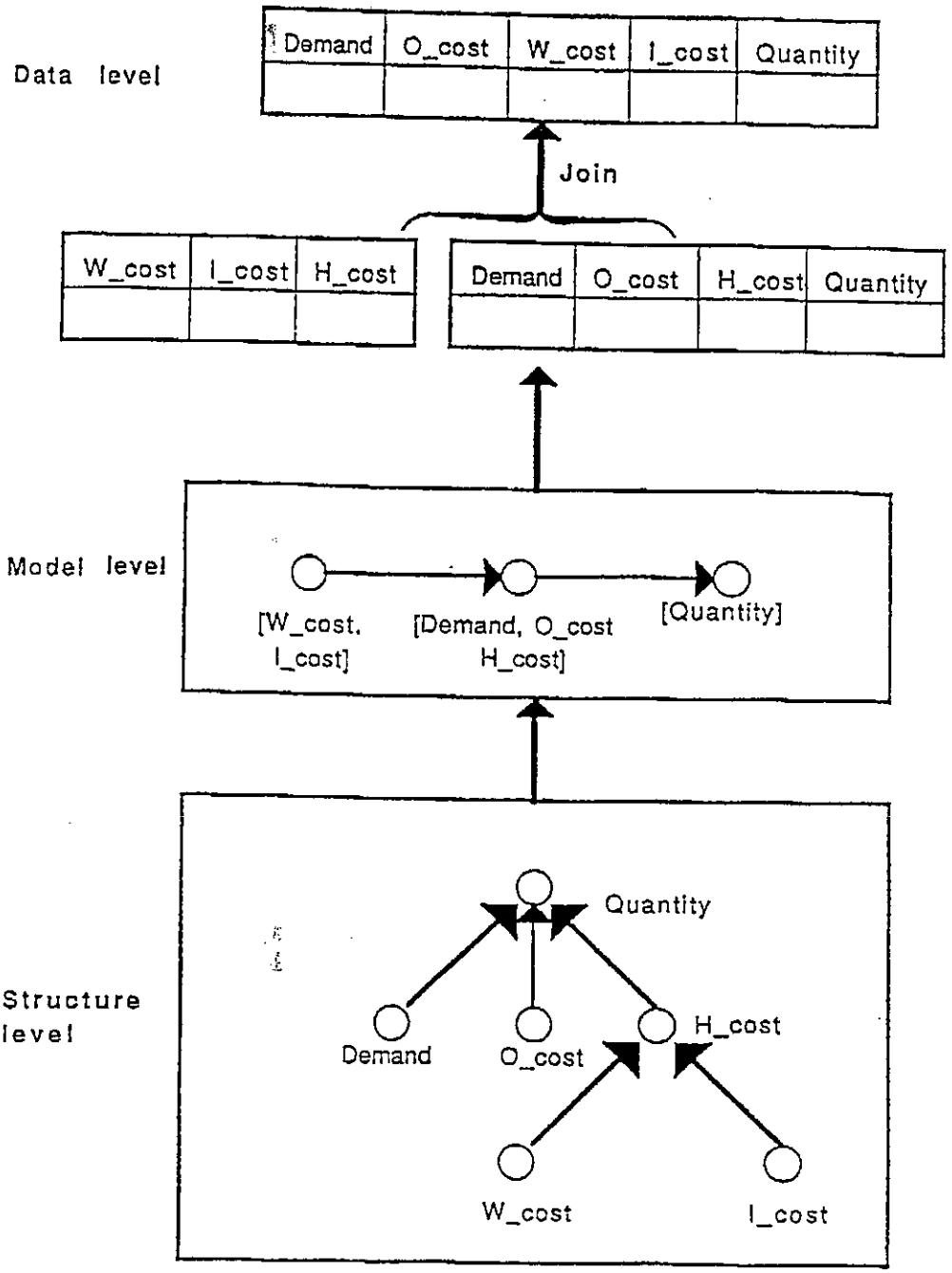

FIGURE 3. Three levels of model integration.

able for solving the problem, the second stage of model selection is activated. It allows various composite models to be examined based on the assumptions, compatibility, and other integrity constraints of their member models. If the user is not satisfied with the constructed model, then the process may be repeated. Figure 4 illustrates the process. In a semi-automatic system, the identification

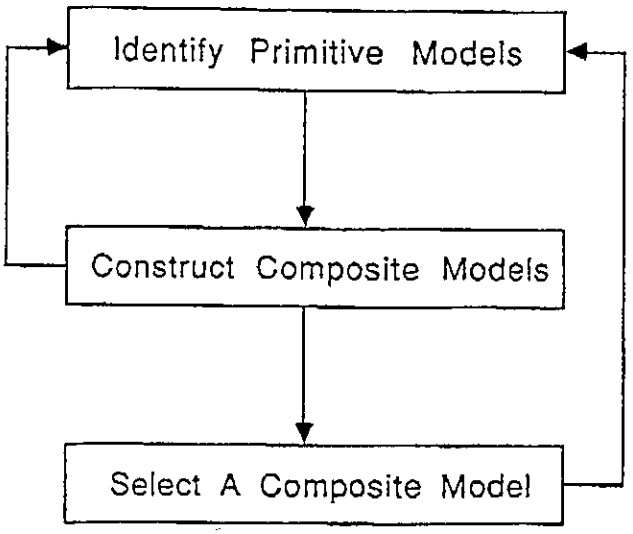

FIGURE 4. Process for model formulation.

and selection of primitive models may be handled by the system, but the selection of the formulated model is made by the user.

Generic Reasoning Strategies

In some situations, identification and scheduling of primitive models are interwoven. In this case, there are three generic strategies that may be employed for model integration: forward reasoning, backward reasoning, and bidirectional reasoning. Each strategy has its advantages and drawbacks. Heuristics may also be used to increase the efficiency of the process. The generic strategies described in this section are based on exhaustive searches. Heuristic functions will be presented in the next section.

The forward reasoning strategy requires the system to start from the initial state and search proper candidates based on the available information. Therefore, it is also called data-directed reasoning. This strategy is appropriate when the goal state is complex but the amount of initial information is relatively small. The algorithm described below applies this strategy to model integration. The queue produced by this algorithm is the master plan for converting the initial state that contains all available inputs, INITIAL, to the goal state that includes the desired output information, GOAL.

```
REPEAT
1. Find an operator, OP, that can convert a state, IN, to another state,
   OUT, where IN is a subset of the initial state, INITIAL;
2. Add OP to a queue and define a new initial state,

   NEWSTATE = (INITIAL U OUT) - IN;
```

```
3. Set INITIAL = NEWSTATE
UNTIL GOAL ⊆ INITIAL.
```

The backward reasoning strategy is output-directed, which requires the system to work backward from the goal state, using facts in the data base to prove it. It works well when the goal state contains a relatively limited number of outputs, and the initial state includes a large amount of input information. A backward model integration process results in a stack which represents a master plan.

```
REPEAT
1. Find an operator, OP, that converts IN to OUT, where
   OUT ∩ GOAL <> ∅;
2. Add OP to a stack and define a subgoal state,
   SUBGOAL = (IN U GOAL) - OUT;
3. Set GOAL = SUBGOAL
UNTIL GOAL ⊆ INITIAL.
```

Bidirectional reasoning uses forward and backward strategies simultaneously. Its major advantage is that it decomposes a problem into two parts, which can reduce the complexity when the number of nodes at each step grows exponentially with the number of steps that have been taken. The risk of using this strategy is that the two searches may pass each other, resulting in more work than it would have taken for either one.

Because the amount of available information is usually much larger than the amount of desired information in most modeling situations, backward reasoning is more appropriate for developing a master plan for model integration. The other two strategies, however, may be useful for explaining the modeling process and helping the user understand the causal relationships between inputs and outputs.

Instead of defining the whole set of desired information as the goal state, a modified backward reasoning strategy, called difference elimination, focuses on the difference between the desired state and the initial state. Since part of the desired information may be readily available from the data base, ignoring the information available in the initial state can simplify the goal state and hence reduce the complexity of the reasoning process. In this case, the model integration process can be defined as a process by which the difference between the desired information and the available information can be completely eliminated. The reasoning process for difference elimination can be described as follows:

```
REPEAT
```

1. Set the goal state as the difference between the desired information and the available information,

   GOAL = DESIRED - INITIAL;

2. Find an operator, OP, that converts IN to OUT, where

   OUT ∩ GOAL <> ∅;

3. Add OP to a stack and define a subgoal, SUBGOAL = (IN U GOAL) - OUT;

4. Set GOAL = SUBGOAL - INITIAL;

UNTIL GOAL = [].

Heuristic Search

Although the difference elimination process may reduce the complexity of the goal state, it still relies on exhaustive search. To improve the efficiency of the search process further, two techniques may be used in model integration: macro-operators and heuristic evaluation functions.

Macro-Operators

A macro-operator is a sequence of primitive operators. Because the sequence is predetermined, there is no need for search. In fact, the major purpose of model integration is to design a macro-operator that can be used to solve a particular problem. The issue, then, is how to use macro-operators to improve the efficiency of modeling. A macro-operator is both the result of a modeling process and a tool for modeling. A master plan developed for solving a previous problem can be saved as a macro-operator for later use. When a new problem includes the previously solved problem as a subproblem, the macro-operator previously developed can be retrieved and fitted directly into the master plan for solving the new problem. From this perspective a model integration process is also a learning process by which macro-operators are learned.

Using macro-operators involves a trade off between model storage costs and modeling costs. On the one hand, the extensive use of macro-operators needs a large amount of computer memory for storage. On the other, lack of macro-operators needs extensive search in the modeling process. Therefore, an important issue here is the extent to which macro-operators should be used.

In general, there are two criteria for determining proper use of macro-operators: functional dependency and frequency of use. A model is defined as functionally dependent on another model if at least one input of the former is among the output of the latter. If there exists a set of functionally dependent models, then they may be grouped into a macro-operator.

Determining functional dependency is also important for model scheduling. Model B must be scheduled before model A when model A is functionally dependent on model B. In addition, a set of functionally dependent models may

result in a cycle, which traps the system into an infinite loop. In this case, mechanisms for detecting and resolving loops must be applied.

Another factor to be considered is the frequency of model utilization. If a particular sequence of operators is used frequently, then it may be appropriate to save them as a macro-operator. Otherwise, it may be unnecessary. For example, if the holding cost model and the EOQ model shown in Fig. 3 are usually used together in an organization, then it may be stored as a macro-operator consisting of these two models to create a mapping from [Demand, O_cost, W_cost, I_cost] to [Quantity].

Heuristic Function

In addition to macro-operators, heuristic functions may be used to guide a search process. The major purpose of a heuristic function is to estimate how close a particular state is to the goal state so that the system can select the best search direction accordingly. In general, a heuristic evaluation function provides a numerical estimation of the promise of a state, which may depend on the criteria used in the function, the description of the goal, and the information gathered by the search up to that point.

Intuitively, there are several criteria that may be used to estimate the promise of a state, such as model applicability, machine capacity, modeling costs, and user preference. Model applicability uses context-dependent measures to evaluate the applicability of a model to a particular problem. For example, when the problem is identified as an inventory problem, then the EOQ model may have a higher value when compared with a capital budgeting model.

By using machine capacity or modeling costs as the major criterion requires that a heuristic evaluation function be used to assess the anticipated machine capacity or modeling costs for each state before selecting the best direction for further exploration. It focuses on developing efficient or cost-effective models. User preference is a subjective measure of model applicability. Its goal is to develop the most satisfactory model for a particular user. Although most of these measures are related to model construction, they are more appropriate for evaluating the formulated master plan than for directing its construction.

A more proper criterion for developing a master plan for model integration is the distance between the resulting subgoal state and the ultimate goal state after applying the model. Since each state represents a set of data attributes, the distance between two states can be defined as the difference between the number of items contained in those two states. For example, the distance between state A and state B in Fig. 5 is 1, because the arc reduces the number of items in the goal state from 5 to 4.

Because the difference elimination process is basically a backward reasoning process, the system should pursue an operator that results in a state far from the goal state. In other words, the system will select the model that eliminates the

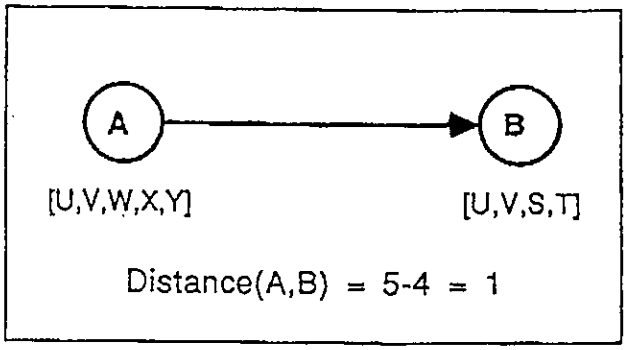

FIGURE 5. Distance between two states.

largest number of items from the desired goal state. This presumably would develop a master plan in the shortest time with a minimum number of basic models.

One point that needs to be addressed is that cycles must be eliminated during the planning process. A cycle is a situation in which an input of model i is the output of another model j that includes at least one output of model i as its input. The existence of a cycle can lead to an infinite loop in the reasoning process and hence must be detected and removed. By taking advantage of the distance-based heuristic evaluation function, the reasoning process for model integration is modified as follows:

```
REPEAT
1. Set the goal state as the difference between the desired information
   and the available information,

   GOAL = DESIRED - INITIAL;

2. Find all operators, OPS, each of which results in an OUT state, where OUT
   ∩ GOAL = 0;
3. Check functional dependencies and cyclicity to exclude the operators
   functionally dependent on others and remove loops;
4. Determine the resulting subgoal for each of the remaining operators
   that converts IN to OUT, SUBGOAL = (IN ∪ GOAL) - OUT;
5. Calculate the distance for each of the possible subgoals,
   Distance(GOAL,SUBGOAL) = Item(GOAL) - Item(GOAL) - Item(SUBGOAL);
6. Select the operator with the longest distance, OP, which converts IN to
   OUT;
7. Add OP to a stack and define a subgoal, SUBGOAL = (IN ∪ GOAL) - OUT;
8. Set GOAL = SUBGOAL - INITIAL;
UNTIL GOAL = [].
```

TIMMS: AN IMPLEMENTATION

On the basis of previously described reasoning mechanism, a model management system called TIMMS has been implemented in PROLOG (see Liang

[1988] for a detailed description of the system). The system includes a model construction module that constructs composite models by integrating the models stored in the model base, a model utilization system that facilitates the use of the constructed models, and an inference engine that links the model base, data base, and knowledge base. Figure 6 shows the architecture of the system.

TIMMS uses a SQL-like language to interact with the user. The user specifies the desired information, and the system retrieves it from the data base if it is available. Otherwise, the system searches the model base to find a model that generates the information. If there exists a model capable of producing the information, then the system activates and executes the model to provide the output to the user. If no existing model is found in the model base, then the system activates the model integration mechanism to construct a master plan for model integration and executes the plan to produce the output upon request by the user.

For example, we assume that the system maintains a data base containing ordering and holding costs and annual sales up to 1990 and a model base containing an inventory control model for calculating the EOQ and three sales forecasting models for predicting sales for 1991. If the user needs to know EOQ for 1991, the system cannot find the information in the data base. Therefore, it searches the model base to find the inventory control model. To execute the model, however, the system finds that a predicted sales for 1991 is necessary. The system again finds that the information may be generated by a sales forecasting model. The model integration mechanism then constructs a master plan that integrates a sales forecasting model and the inventory control model. Figure 7 illustrates a sample session. The user may ask for an explanation of the composite model. Figure 8 shows a sample explanation screen.

Appendix 1 lists a sample PROLOG implementation of the difference elimination process for developing the master plan for model integration. The PROLOG implementation shows the following modules: (1) a main planning module for defining goals and subgoals, (2) a difference determination module for finding the difference, (3) a dependency checking module, and (4) a heuristic evaluation function for selecting the best operator.

Since the modeling process is a cooperative process between the user and the system (see Fig. 1), the system should allow the user to examine the details of the models and decide whether the model is an acceptable one. In the selection process, the user may ask the system to go from the master plan down to the structure, specification, or even program level shown in Fig. 2 to check assumptions, integrity constraints, and data formats. If a certain plan is not acceptable or if there is a violation of the assumptions of member models, then the user may reject the proposed model to examine the next available alternative.

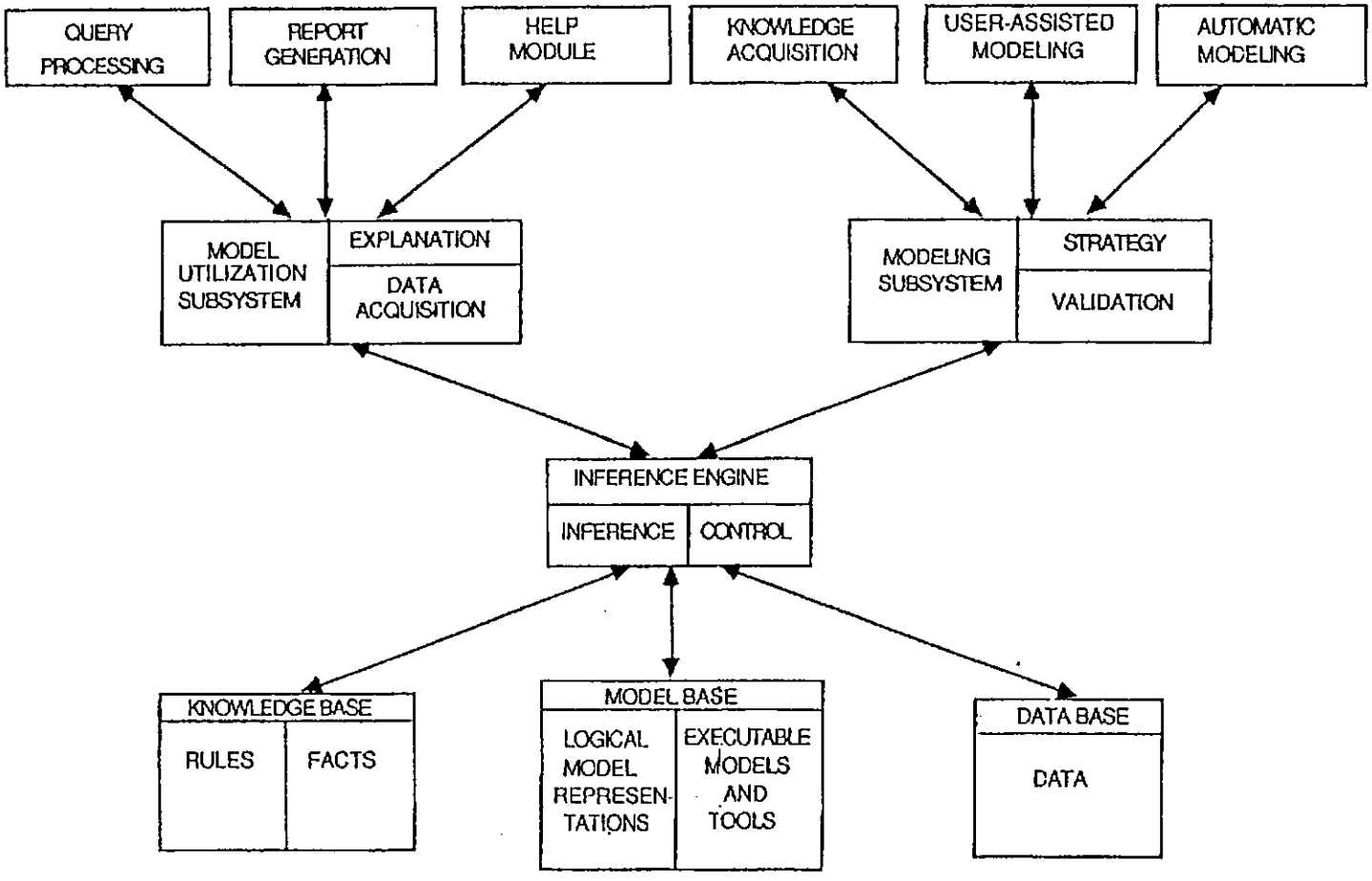

FIGURE 6. Architecture of TIMMS.

```
**********************************************************
**          TIMMS: QUERY PROCESSING SUBSYSTEM           **
**********************************************************
Please specify the information you need:
        OUTPUT: eoq
        WHERE:
            product = a
            year = 1991

Please wait while checking the database
'eoq for a for 1991' is not available in the database

I am checking the model base

'demand for a for 1991' is needed but not available in the database
Could you provide it (y/n)? n

'price of a' is needed but not available in the database
Could you provide it (y/n)? y

Please enter the value: 10

-------------------------------------------------------
                    MY SUGGESTIONS
-------------------------------------------------------
There are three ways to produce the requested output.

The first is:  Integrating model 'M1' and model 'M2'

Execute 'M1' to generate 'eoq for a for 1991'

The execution of 'M1' needs the following three inputs:
        -- holding_cost of a
        -- ordering_cost of a
        -- demand for a for 1991
The database has ..... holding_cost of a = 5
The database has ..... ordering_cost of a = 20

'demand for a for 1991' can be generated by executing model 'M2'

The execution of 'M2' needs the following 1 input:
        -- price of a
You provided ..... price of a = 10

Do you want to execute this model (y/n)? y

** eoq for a for 1991 = 12

More suggestions? (y/n)? n

-------------------------------------------------------
                       THANK YOU
-------------------------------------------------------
```

FIGURE 7. A sample session.

CONCLUDING REMARKS

Developing a master plan is a key step in model integration. In this paper we have discussed several key issues. First, a hierarchy of model abstraction has been described. Then, various reasoning strategies have been presented. They

HELP MESSAGES

The model is an integration of the following models:

** M1 **

M1 is a model for computing the economic order quantity. It needs three kinds of inputs: holding cost, ordering cost, and demand for the year. The formula for computing the order cost quantity is as follows:

$$eoq = \left(\frac{2 * \text{ordering cost} * \text{demand}}{\text{holding cost}}\right)^{1/2}$$

** M2 **

M2 is a demand forecasting model which forecasts future demand according to a pre-defined demand function, as follows:

demand = 2000/unit price

<<PRESS ANY KEY TO CONTINUE>>

FIGURE 8. A sample explanation screen.

include forward, backward, and bidirectional strategies and a modified backward strategy called difference elimination. A heuristic evaluation function for improving the reasoning efficiency has also been developed. Finally, TIMMS, an implementation in PROLOG, is briefly described.

Since reasoning for model integration is critical to the successful development of MMS, further research that investigates the following issues is required before a practical system can be developed. First, how can various levels of abstraction be integrated into a single system? Efficient algorithms must be developed to link representations at different levels. Second, how can operators and macro-operators be determined? In this paper, the issue has been briefly discussed. Two criteria, functional dependency and frequency of use, have been described, but more detailed discussions are needed. Third, how can heuristic evaluation functions be developed and evaluated? A measure of the distance between two states and a distance-based heuristic evaluation function have been presented. This, however, should not be considered the only method. Further research is needed to explore alternative measures of the distance between states and to evaluate various measures to find the most appropriate one.

APPENDIX 1: A PROLOG IMPLEMENTATION OF THE REASONING MECHANISM

```
model_integration:-
     write('Output attributes = '), read (GOAL),
     write('Input Attributes = '), read (START),nl,
     write('** Modeling goal = '), write(GOAL),nl,nl,
     modeling(START,GOAL,START,ACTS),nl,
     write('*** The master plan is to execute the following'),
     nl,write('models sequentially:'),nl,nl,
     write(ACTS).

modeling(STARTSTATE,GOAL,STARTSTATE,[]):-
  satisfy(STARTSTATE,GOAL),!.

modeling(STARTSTATE,GOAL,ENDSTATE,ACTS):-
  majordiff(STARTSTATE,GOAL,DIFF),
  findall_acts(STARTSTATE,DIFF,ALLACT),
  write('all possible candidates = '), write (ALLACT),nl,
  check_dependency(ALLACT,RESTACT),
  write('Independent candidate = '),write(RESTACT),nl,
  heuristic(RESTACT,ACT,V),
  write('Selected action by a heuristic = '),write(ACT),nl,
  subgoal(STARTSTATE,DIFF,ACT,START1,SUBGOAL),
  nl,write('** New subgoal = '),write(SUBGOAL),nl,nl,
  modeling(START1,SUBGOAL,_,ACT1),
  append(ACT1,ACT,ACTS).

satisfy(STARTSTATE,GOAL):-
  subset(GOAL,STARTSTATE).

check_dependency([X],[X]).
check_dependency(ALLACT,RESTACT):-
  all_in(ALLACT,TIN),
  del_prereq(ALLACT,TIN,RESTACT).

all_in([],[]).
all_in([a(ACT,V)],TIN):-
   model(ACT,TIN).
all_in([a(ACT,V))|REST],TIN):-
   model(ACT,IN,OUT),
   all_in(REST,TIN1),
   union(IN,TIN1,TIN).

del_prereq([],[]).
del_prereq([a(ACT,V)|REST],TIN,RESTACT):-
   model(ACT,IN,OUT),
   subset(OUT,TIN),
   del_prereq(REST,TIN,RESTACT).
del_prereq([a(ACT,V)|REST],TIN,[a(ACT,V)|RESTACT]):-
   del_prereq(REST,TIN,RESTACT).
```

```
/* Determining major difference between START and GOAL */

majordiff(_,[],[]).
majordiff(START,[X|R],Z):-
  member(X,START),!,
  majordiff(START,R,Z).
majordiff(START,[X|R],[X|Z]):-
  majordiff(START,R,Z).

/* Submodels required for bridging the difference between the starting
   state and the goal */

findall_acts(STARTSTATE,DIFF,ALLACT):-
    model(ACT,IN,OUT),
    intersection(OUT,DIFF,CONT),
    CONT\ = [],
    listlength(CONT,L1),
    majordiff(STARTSTATE,IN,NEG),
    listlength(NEG,L2),
    L is L1 - L2,
    assertz(stack(a(ACT,L))),fail;
    assertz(stack(a(end,[]))),
    collect(ALLACT).

collect(ALLACT):-
  retract(stack(X)),!,
  (X == a(end,[]),!, ALLACT = [];
  ALLACT = [X|REST],collect(REST)).

heuristic([a(ACT,V)],[ACT],V).
heuristic([a(ACT1,V1)|REST],[ACT1],V1):-
  heuristic(REST,[BestACT],VMax),
  V1 >= VMax.
heuristic([a(ACT1,V1)|REST],[BestACT],VMax):-
  heuristic(REST,[BestACT],VMax),
  V1 < VMax.

/* finding subgoals */

subgoal(STARTSTATE,DIFF,[ACT],START1,SUBGOAL):-
  model(ACT,IN,OUT),
  union(STARTSTATE,OUT,START1),
  union(DIFF,IN,GOAL),
  Majordiff(START1,GOAL,SUBGOAL).

/* Utilities */

append([],L,L).
append([A|X],Y,[A|Z]):-append(X,Y,Z).

member(X,[X|_]).
```

```
member(X,[_|Y]):-member(X,Y).
subset([A|X],Y):-member(A,Y),subset(X,Y).
subset([],Y).

Intersection([],X,[]).
intersection([A|R],Y,[A|Z]):-
  member(A,Y),!,intersection(R,Y,Z).
intersection([A|R],Y,Z):-intersection(R,Y,Z).

listlength([],0).
listlength([H|R],L):-
   listlength(R,L1),
   L is L1 + 1.

union([],X,X).
union([H|T],Y,Z):-member(Y,Y),!,union(T,Y,Z).
union([H|T],Y,[H|Z]):-union(T,Y,Z).
```

REFERENCES

Blanning, R. W. 1982, A Relational Framework for Model Management in Decision Support Systems, *DSS-82 Trans.*, 16–28.

Blanning, R. W. 1985a, A Relational Framework for Join Implementation in Model Management Systems, *Decision Support Sys.*, 1:69–81.

Blanning, R. W. 1985b, A Relational Framework for Assertion Management, *Decision Support Sys.*, 1:167–172.

Blanning, R. W. 1986, An Entity-Relationship Approach to Model Management, *Decision Support Sys.*, 2:65–72.

Bonczek, R. H., Holsapple, C. W., and Whinston, A. B. 1981, *Foundations of Decision Support Systems*, New York: Academic.

d'Alessandro, P., Dalla Mora, M., and De Santis, E. 1989, Issues in Design and Architecture of Advanced Dynamic Model Management of Decision Support Systems, *Decision Support Sys.*, 5:365–377.

Darden, L. 1987, Viewing the History of Science as Compiled Hindsight, *AI Mag.*, 8:33–41.

Dempster, M. A. H., and Ireland, A. M. 1989, Object-Oriented Model Integration in MIDAS, *HICSS-22 Proc.*, 3:612–620.

Dolk, D. R. 1986, Data as Models: An Approach to Implementing Model Management, *Decision Support Sys.*, 2:73–80.

Dolk, D. R., and Konsynski, B. R. 1984, Knowledge Representation for Model Management Systems, *IEEE Trans. Software Eng.*, SE-10:619–628.

Dutta, A., and Basu, A. 1984, An Artificial Intelligence Approach to Model Management in Decision Support Systems, *IEEE Computer*, 17:89–97.

Elam, J. J., Henderson, J. C., and Miller, L. W. 1980, Model Management Systems: An Approach to Decision Support in Complex Organizations, Proceedings of the First International Conference on Information System, Philadelphia, PA: University of Pennsylvania.

Geoffrion, A. M. 1985, Structured Modeling, report, Graduate School of Management, University of California, Los Angeles.

Geoffrion, A. M. 1987, Introduction to Structured Modeling, *Management Sci.*, 33:547–588.

Geoffrion, A. M. 1989, Reusing Structured Models via Model Integration, *HICSS-22 Proc.*, 3:601–611.

Kimbrough, S. O. 1986, A Graph Representation for Management of Logic Models, *Decision Support Sys.*, 2:27–37.

Konsynski, B., and Dolk, D. R. 1982, Knowledge Abstractions in Model Management, *DSS-82 Trans.*, pp. 187–202.

Korf, R. E. 1985, Macro-Operators: A Weak Method for Learning, *Artificial Intell.*, 26:35–77.
Korf, R. E. 1987, Planning as Search: A Quantitative Approach, *Artificial Intell.*, 33:65–88.
Krishnan, R. 1990, A Logic Modeling Language for Automated Model Construction, *Decision Support Sys.*
Liang, T. P. 1985, Integrating Model Management with Data Management in Decision Support Systems, *Decision Support Sys.*, 1:221–232.
Liang, T. P. 1986, A Graph–based Approach to Model Management, Proceedings of the Sixth International Conference on Information Systems, San Diego, CA, 136–151.
Liang, T. P. 1988, Development of a Knowledge-Based Model Management System, *Operations Res.*, 36:849–863.
Liang, T. P., and Jones, C. V. 1988, Meta-Design Considerations in Developing Model Management Systems, *Decision Sci.*, 19:72–92.
Muhanna, W. A., and Pick, R. A. 1986, A Systems Framework for Model Management, Working Paper, University of Wisconsin at Madison.
Newell, A., and Simon, H. 1972, *Human Problem Solving*, Englewood Cliffs, N.J.: Prentice Hall.
Pan, S., Pick, R. A., and Whinston, A. B. 1986, A Formal Approach to Decision Support, in S. K. Chang (ed.), *Management and Office Information Systems*, New York: Plenum.
Polya, G. 1957, *How to Solve It*, Garden City, N.Y.: Doubleday.
Sacerdoti, E. 1974, Planning in a Hierarchy of Abstraction Spaces, *Artificial Intell.*, 5:115–135.
Simon, H. A. 1981, *The Science of the Artificial*, Cambridge, Mass.: MIT Press.
Simon, H. A. 1983, Search and Reasoning in Problem Solving, *Artificial Intell.*, 21:7–29.