Decision Support Systems 10 (1993) 213–233
North-Holland

Supporting complex real-time decision making through machine learning

Alok R. Chaturvedi
Purdue University, West Lafayette, IN, USA

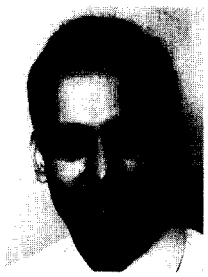
George K. Hutchinson
and Derek L. Nazareth
University of Wisconsin-Milwaukee, Milwaukee, WI, USA

This paper presents FMS-DSS, a system for supporting complex, real-time decision making in the FMS scheduling and control domain. FMS-DSS differs from traditional DSSs in that it can acquire scheduling and control knowledge from historical data comprising prior decisions. This knowledge is applied to support subsequent decision making. It manages complexity through hierarchically structuring the user's objectives, and can deal with noise in the form of missing, inaccurate, or erroneous data. Results indicate that a machine learning based approach can provide effective support for repetitive real-time decision making and outperform static scheduling rules.

Keywords: Real-time decisions; FMS scheduling; Machine learning; DSS.

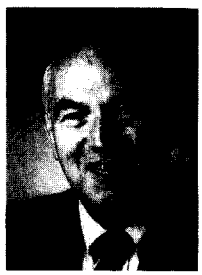
1. Introduction

This paper describes a decision support system for real-time control of manufacturing systems. The concept of a decision support system (DSS) to assist human decision makers in semi-structured tasks has been articulated several years ago [21]. Though there has been much effort in developing DSS, surveys of applications indicate that the bulk of activity has been directed at systems that do not involve any real-time control [11]. Using Sprague and Carlson's extension [44] to the Gorry and Scott Morton framework [14], it would appear that decisions made on a dynamic basis would be classified as operational performance and operational control decisions, and characterized as semi-structured to unstructured. Examples of these decisions include dispatching, con-

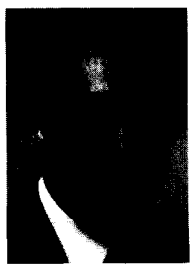


Alok R. Chaturvedi is Assistant Professor of Management Information Systems in the Krannert Graduate School of Management at Purdue University. His research interests include artificial intelligence applications in manufacturing, conceptual aggregation and machine learning applied to decision making, and cooperative computing. He has a B.Sc. degree in Mechanical Engineering from Birla Institute of Technology, Ranchi, India, and M.S. and Ph.D. degrees in Management Information Systems from the University of Wisconsin-Milwaukee. He is a member of ACM, AAAI, ORSA, and TIMS.

Correspondence to: A.R. Chaturvedi, Krannert Graduate School of Management, Purdue University, West Lafayette, IN 47907, USA. (317) 494-9048. alok@zeus.mgmt.purdue.edu.



George K. Hutchinson is Research Professor of Management Information Systems at the University of Wisconsin-Milwaukee. He is also a Professional Engineer with a BS in Mechanical Engineering from the University of Maine, an MS in Mechanical Engineering from Carnegie Mellon University, and a Ph.D. in Business from Stanford University. He has published over 40 papers-and is a recognized authority in simulation, flexible manufacturing systems, and online control systems. Dr. Hutchinson is affiliated with SME, ORSL, ORSA, ACM, IEEE, and has been part of the Executive Council of the Society of Management Information Systems. He is a representative to IFIP WG 5.7 Technical Committee for Automation of Production Planning and Control, and has served on several committees of the National Academy of Science.



Derek L. Nazareth is Assistant Professor of Management Information Systems at the University of Wisconsin-Milwaukee. He holds a Ph.D. in Management from Case Western Reserve University. His previous work appears in *IJMMS, Knowledge Acquisition, INFOR*, and *Journal of Intelligent Manufacturing*. His current research interests include knowledge-base verification, decision support systems, and flexible manufacturing systems. He is a member of AAAI, ACM, IEEE Computer Society, and TIMS.

tinuous investment advising, manufacturing systems scheduling and control. In the increasingly competitive and automated environment that organizations currently face, it is expected that effective performance in these decisions will assume critical importance for organizations. However, there appears to be a dearth of strategies for constructing Systems that provide effective support for real-time decisions.

A closer examination of the nature of these real-time decisions reveals the difficulties in providing effective support using traditional DSS approaches. Real-time decisions differ from conventional decisions in several important aspects. First, they involve continual scanning of the environment. Second, they are made on an extremely repetitive basis and address a short-time horizon only. Third, hard real-time systems require that the decision be made in a short-time frame. Additionally, the long-term effects of these decisions are difficult to anticipate or control. Thus for example, in the dispatching situation, the tendency to use the nearest available taxicab, though intuitively appealing, may lead to an unbalanced workload. Or in the manufacturing example, the strategy of releasing the job with the earliest due date may create significant in-process inventories. As is the case with most decisions, real-time systems involve a number of conflicting objectives. Optimal performance on any objective is not easily realized on account of the inability to closely relate overall performance to individual decisions made during the period under observation.

The differences cited pose significant barriers to the effective application of traditional DSS development strategies to real-time situations. Continual scanning introduces the need to deal with large volumes of data. In some cases, the amount of raw data available may pose a problem in terms of its collection and management. Coupled with this is the fact that data may be collected from several sources, with uncertain degrees of reliability. Traditional database management strategies, though not theoretically limited by size, tend to perform poorly when dealing with extremely large volumes of data. In addition, models that depend on this data for input, now have to be robust enough to deal with missing, inaccurate, or erroneous data. Moreover, significant computational resources may have to be devoted to the processing and maintenance of this data.

The need to make decisions on a repetitive basis and under extreme time pressure also presents some problems for effective support, if the decisions are to be made by humans. The tendency to employ a satisficing approach to reduce the search space may lead to an examination of fewer alternatives, fewer criteria, or lower acceptance standards, thereby compromising decision quality. Automating the decision represents an appealing alternative, in that the decision quality may not be compromised, but entails reduced human intervention and greater inflexibility. It is instead suggested that a symbiotic partnership between man and machine be adopted, whereby significant preprocessing is done by the computer thereby simplifying the decision task for the human. Moreover, it may be of some advantage to store information on prior decisions so that the decision maker does not have to re-evaluate recurring decisions anew. The information can be profitably employed when faced with a decision-making situation that has been encountered and successfully dealt with in the past. It is expected that a significant fraction of real-time decisions can be served adequately using prior decision information. Given the change in the role of the DSS, the support provided will necessarily be different.

Alternative approaches to traditional decision support for real-time systems include the use of control theory, queueing theory, and heuristic procedures. Control theory employs rigorous mathematical formulations and is useful in the context of easily interpretable systems. Its formulations tend to involve continuous functions representing inputs, outputs and system states. The class of decisions being examined is better characterized as discrete event systems, and the complexity of mathematics required to formulate an accurate control model usually prohibits effective application. Queueing theory, and stochastic optimization techniques in general, tend to view system entities as centers that serve a continuous job stream described by probability distributions. While they provide reasonably accurate estimates of system outputs, they provide estimated mean values only, and are prone to intractability for large systems. Heuristic procedures have been devised as an alternative to the rigorous mathe-

matical formulations, and seek to achieve specific goals or objectives. The bulk of heuristics employ simple and relatively intuitive rules for decision making. Thus for example in the dispatching case, the nearest available server selection rule appears to minimize wait time. While this is a relatively intuitive strategy, it may not achieve this objective at a global level, especially if new requests originate from areas just vacated by servers. There are more powerful heuristics developed for real-time decision making, typically directed at improving performance on specific objectives. However, their use is not as widespread, and they generally entail a significant degree of added complexity. Most heuristic procedures are devised to address a single objective or goal, whereas frequently the decision maker would like to address multiple, and generally conflicting, goals.

As an alternative to the traditional real-time approaches, this research employs machine learning to deal with system complexity and noise in data. Machine learning provides three major advantages for support of real-time DSS. First, it allows for an explication of the underlying system not generally available through other strategies. The complexity of large real-time systems precludes the formulation of accurate mathematical models that effectively capture all structural and procedural details. Instead, exploration of the relationship between decisions and their outcomes appears more practical. Statistical procedures to explicate this relationship have been suggested. Violations of assumptions and erroneous classification make statistical techniques suspect though. Second, it permits the reduction of large volumes of possibly noisy data into compact relevant data sets. The need to deal with inconsistency in the data reinforces the use of techniques that eliminate the extreme dependence on accuracy. Learning techniques involving abstraction are useful in this context as they permit reduction of voluminous information, and the elimination of extreme dependence on data reliability. Third, it provides a rich source for assisting subsequent decisions, based on prior experience. Given that most real-time decisions are recurrent and are to be made under some time pressure, the ability to provide an instant decision instead of reasoning from first principles offers significant advantage.

This paper presents FMS-DSS, a decision support system for scheduling and control of a flexible manufacturing system (FMS). An FMS is a production unit capable of producing a range of products. It consists of workstations (machine tools or other equipment for fabrication, assembly, or treatment) linked by a materials handling system to move parts from one station to another. In addition, it operates as an integrated system under computer control. Decision making in an FMS environment is characterized by complexity and uncertainty. The complexity can be attributed to the massive quantity of data needed to describe the status of the system, coupled with the vast flexibility in operation. The strategy employed by FMS-DSS uses a combination of case-based and conceptual clustering learning techniques. The new learning strategy, termed goal-directed conceptual aggregation (GDCA), captures and stores information on prior decision making contexts, which is then used to supplement domain knowledge. The use of an aggregation mechanism leads to reduction of the amount and uncertainty of data needed for decision making. Given the repetitive nature of decisions and the short-time frame for decision making, it is desirable to make use of knowledge gleaned from successful prior decisions.

FMS-DSS is implemented on an IBM PS/2 Model 80 in the GoldWorks knowledge engineering environment. Results indicate that this form of DSS can be successfully employed in supporting real-time decisions, through the use of machine learning techniques to reduce the complexity and uncertainty of the problem. It is important to note that the role of FMS-DSS is to assist real-time decision making, and not to perform real-time learning. Learning is acknowledged to be a complex process, and generally involves time and effort. Though FMS-DSS does learn in a dynamic manner through constant adjustment, the acquisition and refinement of knowledge occurs over an extended period of time. Domain knowledge gleaned through the learning process is available for real-time decision making.

The organization of the rest of the paper is as follows. Section 2 provides an overview of machine learning techniques and surveys their application in decision support. The application task is described in section 3, including a detailed description of the model for the FMS under study. Section 4 describes the architecture of FMS-DSS

for decision support using machine learning. The operation of FMS-DSS is presented in section 5. Section 6 presents an example session of decision making with FMS-DSS. Section 7 presents the results of FMS-DSS use, making a strong case for the use of machine learning in complex real-time decisions. Implications for this approach to real-time decision support complete the paper.

2. Overview of machine learning

The need to abstract information in an automated manner from large quantities of data for present and future reference suggests the use of machine learning principles. Machine learning seeks to acquire knowledge about a specific domain from available data in an automated manner. The learning process may be supervised or autonomously guided. Machine learning techniques generally employ a small number of extremely general induction strategies coupled with some basic domain knowledge. The knowledge inferred may involve structural descriptions, procedural explanations, or even discoveries of new domain concepts. Knowledge may be stored as rules, semantic nets, logic predicates or frames. Due to the inductive nature of reasoning involved there is always the possibility of error. Consequently, most techniques allow for self-improvement through disconfirmatory feedback. This section provides an overview of current machine learning techniques and their application in decision support. Learning techniques are classified on the reasoning processes adopted to induce new knowledge.

Early machine learning techniques were directed towards recognizing patterns in data through parametric adjustment, as embodied in perceptrons [37,38] and Boltzman machines [16]; or signature tables, illustrated in Samuel's checkers playing program [39,40]. Though effective for the tasks at hand, they could be applied to a limited set of domains, and the knowledge acquired was not necessarily stable. These techniques have been described as *learning by being told* or *rote learning*.

More general learning strategies were later formulated using descriptions. These included reasoning from confirmatory and disconfirmatory examples [47], induction through discriminant descriptors [29], and concept learning [35,46]. This class of learning techniques adopted more formal approaches for acquiring knowledge and employed richer knowledge structures, as in semantic nets (Winston's program), predicate calculus expressions (AQ11/INDUCE), decision trees (ID3), and acyclic graphs (CONIS). The essential strategy involved creating discriminant knowledge from a training set, which could subsequently be applied to other observations. Several successful systems were developed, including one that could outperform experts in identifying soybean plant diseases [30]. These techniques have been termed *learning by example* strategies.

Further refinements in machine learning have been achieved through the use of concepts, rather than descriptions. Starting from a basic set of concepts in a domain, and general principles of induction, it is possible to create more meaningful higher-level concepts. This approach has been employed in some of the clustering problems [13,31], and have been characterized as *learning by observation* strategies. These learning strategies represent unsupervised learning, and differ from *learning by example* in that negative and positive examples are not required for training purposes.

Other learning strategies involve the formation of new concepts based on their similarity to known concepts, through some measure of similarity between new and existing concepts. These strategies have been termed as *learning by analogy*. Analogical reasoning can be applied within or across domains. These strategies have been applied to legal reasoning [28], metaphors [48], and are based upon work by Carbonell [4]. Surveys of analogical learning appear in [15,20].

Learning strategies that utilize rich domain knowledge and concentrate on individual examples to refine this knowledge have also been devised. These have been characterized as *learning by explanation* strategies. Though they appear to be similar to learning by example approaches, they differ in the scope of background knowledge employed. The knowledge employed in *learning by explanation* strategies includes structural and procedural knowledge in a model that permits incremental refinement based on individual cases. Explanation based learning strategies derive from macro-operator formulation originally employed in STRIPS [12] and are summarized in [10,32,33].

The bulk of learning strategies described thus far work with a well-defined set of heuristics. There is a class of learning techniques that modify these heuristics as the system acquires more knowledge. AM [9] and EURISKO [22,23,24,25] are examples of these methods which have been termed *learning by discovery*.

Other taxonomies for machine learning techniques include structures based on architecture, embodying *connectionist learning* [17], and *genetic learning* [2,18]. Frequency based taxonomies distinguish between *one-shot learning* and *incremental learning*.

Given the need for flexibility and constant evolution of models employed, it would appear that DSS would benefit greatly from some machine learning capability. However, machine learning has only recently been employed for decision support. Examples include the use of discriminant descriptors for inductive learning for business loan evaluation [41]. Machine learning techniques have also been applied to real-time systems, though the applications appear to involve static learning. These include the use of concept learning via decision trees for the control of production processes [3]; clustering techniques

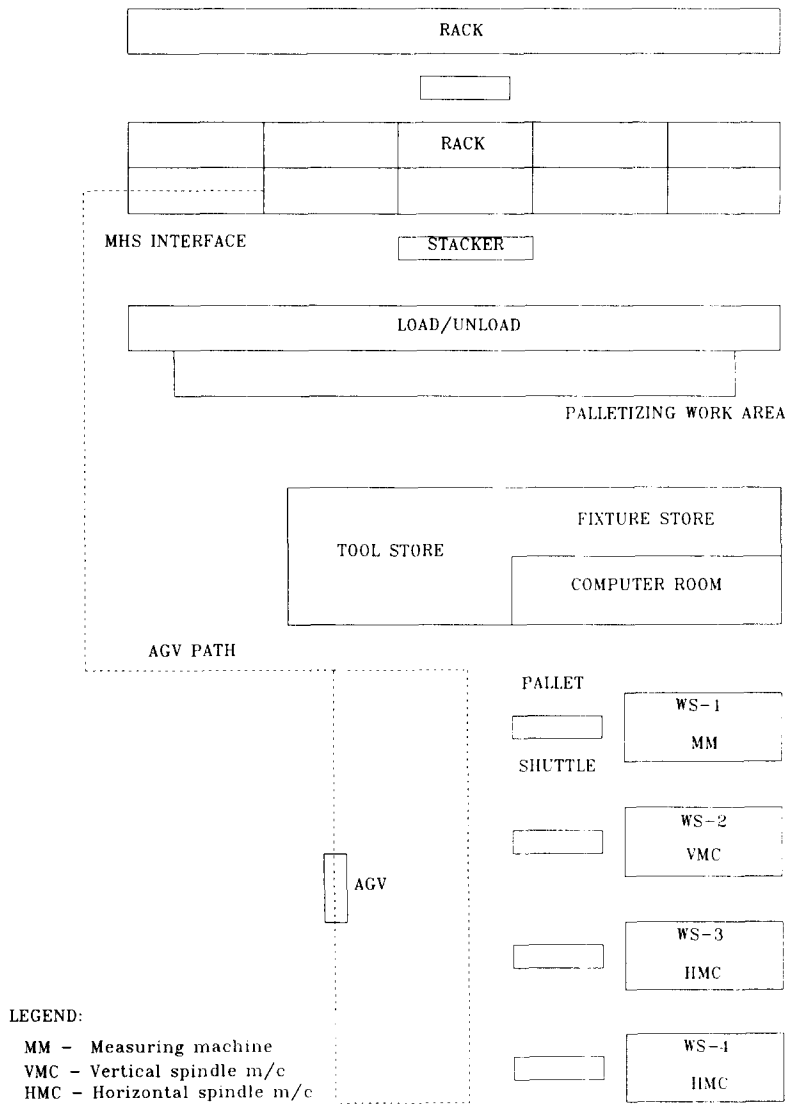


Fig. 1. FMS physical layout.

for production design and control knowledge [8]; and a variety of techniques for scheduling and control of flexible manufacturing systems including decision trees [34], macro-operators [43], nonlinear planning [42], and conceptual aggregation [5]. Machine learning principles have also been applied to model management. Examples include the use of analogical problem solving in model formulation [26,27]. However, this paper is concerned more with the use of machine learning for specific DSS applications, as opposed to generalized DSS architectures.

3. The application task

FMS-DSS supports scheduling and control decisions for a cellular FMS. The problem domain provides significant flexibility in terms of physical design characteristics, manufacturing mission, and decision-making strategies adopted. Physical design considerations cover FMS organization (hierarchical or cellular), FMS scope (facility, shop, or cell), and FMS components (workstations, robots, material handling system, and automated guided vehicles, and their layout). The manufacturing mission includes the number of parts, the production rate and routing for each. This provides the capability to vary work loads, tooling assignments, and production mixes, among others. Job release, part loading, and workstation assignment decisions form part of the scheduling and control process. Traditionally these decisions have employed a variety of heuristics, including simple ones as first come first serve, earliest due date, to more focussed heuristics as shortest processing time, etc.

The domain is defined by the totality of choices available in each category and provides a very large number of possibilities. Given the large scope of the domain, claims for unilateral improvement in FMS operation can at best be specious. Instead, this paper examines a specific problem within the context of FMS scheduling and control. The task selected involves the selection and assignment of parts to workstations in a cellular structure, with a view to achieving specific management goals. The decisions covered by this study include selection of workpieces from the order stream for inclusion within the job stream associated with the cell, and the assignment of workpieces to individual workstations.

3.1. Physical characteristics

FMS-DSS models a planned European factory that employs about 2000 people in the production of industrial equipment sold worldwide. It produces a wide diversity of workpieces in relatively short production runs, with approximately 40% of the output belonging to 8 major part families. These part families range from 200–800 mm cube size. Within families there are part dependent variations in processing times. The probability of occurrence of each part family are defined by a process routing and part family variation combination as specified in the manufacturing mission. The physical layout of the plant is shown in fig. 1. The flow of material starts with the delivery of castings from vendors. The external material handling system (EMHS) delivers the castings to a palletizing work area and then to an interface with the stacker crane. The raw castings stay in the storage rack until required by the scheduler. They are then fetched by the stacker, carried to the internal material handling system (MHS) interface, loaded on the MHS, transported to a workstation in-shuttle, and processed at the workstation. Upon termination of this activity, workpieces are moved to the workstation's off-shuttle, picked up by the MHS, returned to the stacker interface, and placed back in the rack, or if completed, sent to shipping. Further details of the system can be found in [6].

3.2. The simulation model

The simulation model of the physical system includes two stacker cranes, one to service the transfer and storage of raw castings and finished product with the EMHS; and the other to handle the movement of fixtures, castings and workpieces in process within the storage area and between storage and the MHS, which is automatic guidance vehicle (AGV) based. The MHS subsystem of the simulation models includes turnouts for each work station to prevent a loading or unloading AGV from blocking passage. As currently modelled, the MHS has three AGVs, each capable of carrying one workpiece. The interface between the workstations and AGVs is a

rotary shuttle which could accommodate two palletized workpieces. Ideally, the AGV fetches old workpieces and delivers new workpieces while the workstation is processing the current workpiece. Workstations in the system include a measuring system, a vertical machining center, and two horizontal machining centers. All workpieces are palletized on pallets, high precision or regular, to fit an $800 \times 1200 \times 215$ mm envelope. The storage racks are all pass-through, i.e. workpieces can be loaded or unloaded from either side. The first storage rack is for castings and finished product, the second for fixtures and workpieces in process, and the third as a buffer for the palletizing work area where the workpieces are fixtured.

The driving function for the FMS model is the arrival of orders. The corresponding entity is a BATCH, consisting of a number of WORKPIECES, and their technological processing requirements, given by a ROUTING which is an ordered linear list of the OPERATIONS necessary to transform a raw casting into a finished product. The operation specifies the type of WORKSTATION which will perform the operation and its duration. The job mix handled by the system is specified by the arrival rate of batches and the probability of occurrence of each type of routing. The STACKER crane provides the interface between MHS, i.e., the storage RACK and the AGVs. AGVs move on predetermined paths between the interface with the STACKER and the WORKSTATIONS, using zone control. The workstations interface with the MHS by rotary SHUTTLES. Each shuttle has positions for two work pieces and provides for interchange between the AGV and WORKSTATION.

All workpieces in a batch have the same processing requirements specified by the batch routing, though several routings are possible. The maximum number of possible routings is controlled by the decision maker. Each routing specifies a list of operations and the workstation types needed to perform the operation. The routing and variation selected is based on a priori probabilities culled from historical data. The arrival time of each batch is recorded for use in decision making and system performance evaluation. The number of batches that are active in the production mix, i.e., available for processing is also under the control of the decision maker. When the last workpiece of a batch is completed, the makespan for the batch is recorded and a new batch is selected from the backlog of those available, using the specified decision rule.

The choice of decision strategies to be employed for the experiments was based on applicability, degree of use, and amenability to revision. Strategies that were specific to individual objectives at the expense of others were excluded. Likewise, esoteric rules that are not widely employed, or introduce significant complexity in the decision making process were also not included. Strategies that are difficult to revise midstream, or are revised at substantial performance penalties, were also excluded. The decision strategies finally chosen were first come first serve (FCFS), shortest processing time (SPT), earliest due date (EDD), and minimum number of jobs remaining (MJR). The decision strategies were employed in a manner as to achieve specific objectives, such as increasing workstation utilization, minimizing tardiness, and reducing WIP inventories. Clearly, other objectives could have been considered, but in the interests of tractability, the analysis was limited to those mentioned. The problem can now be stated more, concisely.

Problem. Given a cellular FMS configuration, decision making objectives, the current status of the FMS, and a historical database of prior decisions, one must determine an appropriate scheduling decision concerning workpiece introduction into the job stream and subsequent assignment to workstations, so as to achieve the objectives to the extent possible, compile knowledge acquired into a form that could be utilized to improve future scheduling decisions, and permits incremental refinement through subsequent experience.

4. FMS-DSS architecture

FMS-DSS is based on a machine learning strategy termed goal-directed conceptual aggregation (GDCA). GDCA is a dynamic learning mechanism that acquires and refines knowledge about the domain in an incremental manner, thus allowing for redefinition of causal relationships between concepts and reorganization of the domain knowledge structure, thereby reducing the extreme dependence on training observations [5].

GDCA has evolved from the conceptual clustering approach [45], and represents an improved learning strategy in that it creates tighter associations between concepts than traditional *learning by observation* strategies, since all concepts are now related to specific goals. Moreover, it recognizes that there may be some imperfections in the data, and employs semantic, contextual, and temporal aggregations to synthesize meaningful knowledge from possibly problematic data. In addition to GDCA's learning capabilities, it can also function in a decision making role. It works on the premise that an organization's database contains a wealth of information that can be gainfully employed in future decision making. The parallel functions of decision making and knowledge acquisition are assisted through the use of selective aggregation of conceptual information.

The choice of the environment for FMS-DSS development was dictated by the need to manage large amounts of possibly uncertain data, as well as to capture and refine FMS scheduling and control knowledge in a dynamic manner. This suggested the use of a frame-based representation, permitting the creation of declarative and procedural knowledge. The need to interface to a ECSL-based simulation system constrained the search to that of microcomputer-based knowledge environments. Goldworks was selected for its ability to provide these features with relative ease. Key features of this implementation include the development of GDCA algorithm, the representation of background knowledge as taxonomies of frames and rules, the design of control structures for reasoning with incomplete information, the design and implementation of a domain database to be used by GDCA, the user interface, and the simulation model. The integrated system is implemented on an IBM PS/2 80.

Conceptually, FMS-DSS fits well with the generic architecture of DSS formulated by Bonczek et al. [1] and is depicted in fig. 2. Its knowledge system contains information about a specific domain (in this case FMS scheduling), its problem processing system provides a mechanism to manipulate this knowledge (the base GDCA system), and its language system provides an interface that facilitates communication with the user. In addition, FMS-DSS also permits ready access to other external entities, viz. the organization's database, and a simulation model used to validate

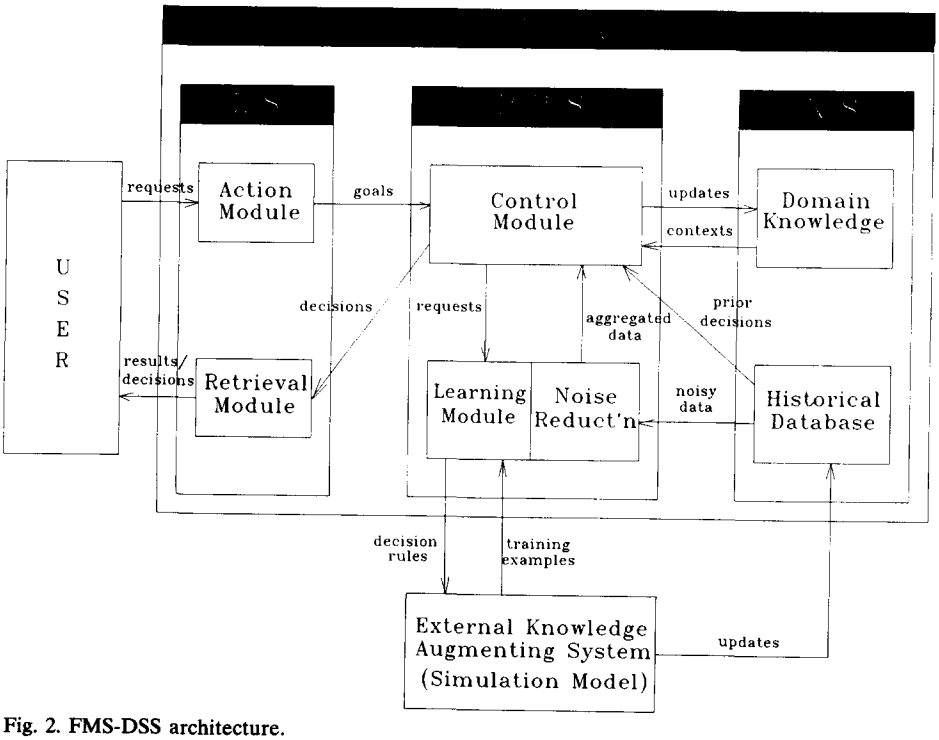


Fig. 2. FMS-DSS architecture.

system performance. Most importantly, FMS-DSS provides a means of refining and embellishing existing domain knowledge, through the use of an active learning module. The structure and functions of the important modules of FMS-DSS are presented in figure 2.

4.1. Language system

The language system in GDCA is currently implemented as a set of imperative verbs. When faced with a decision-making situation, the user specified an action, its target, the objective, any dimensions if relevant, and their directions. In the FMS application, an action would include verbs as SCHEDULE, or ASSIGN; the target could be a BATCH, or JOB; the objectives may include MEET-DUE-DATE, MINIMIZE-WIP; dimensions of interest include AVERAGE-TARDINESS, or MAXIMUM-LATENESS, with directions as IMPROVE, REDUCE etc. In addition, the user may specify multiple objectives, and place different emphasis on each of these objectives. Additionally, the language system supports a set of retrieval commands, in the form of on-line access to the system status and the database, through a QUERY option.

4.2. Knowledge system

The knowledge system comprises a set of conceptual maps about the domain and a database of prior activity. The domain knowledge is provided to the system, and is refined with time, while the database is continually updated.

4.2.1. Domain knowledge

The domain knowledge describes concepts of relevance to the system. These concepts link decision making objectives to decision variables and structural descriptors of the system. Presently, the domain knowledge is structured as hierarchies of frames, samples of which appear in fig. 3.

FMS-DSS's domain knowledge includes entities, aggregation dimensions, and skeletal decisions. Entities represent items of interest in the analysis. They could be physical entities, as a workstation or an operator, or abstract concepts, as a schedule. Anything that is countable or measurable within the system, and is of interest to the user is viewed as an entity. Aggregation dimensions represent the dimensions that can be used to characterize the state of, or otherwise describe an entity. Skeletal decisions provide the goals which seek to aggregate the data as appropriate to the task on hand. These decisions are represented in great generality. More specific goals can be derived through reasoning. The data required by these functions vary in concept and content, and hence, data may be aggregated differently. Skeletal decisions form the basis of capturing decision-making knowledge in FMS-DSS. The taxonomy of domain knowledge is presented in fig. 4.

4.2.2. Historical database

The database stores historical case-based information about decisions, and is employed to narrow the search in regular decision making. Information in the database includes the conditions under which the decision was made, user objectives selected in making the decision, the decision variables selected, and the performance on the chosen objectives.

4.3. Problem processing system

Manipulation and enhancement of the domain knowledge is handled by the problem processing system, which comprises a control module, a noise reduction module for managing uncertainty, and a learning module.

```
(DEFINE-FRAME WORKSTATION
   (:print-name "WORKSTATION"
   (NUMBER-OF-STARTS)
   (TIME-OF-PROCESSING)
   (UTILIZATION)
   (BACKLOG))
```

WORKSTATION Frame

```
(DEFINE-FRAME BATCH
   (:print-name "BATCH"
   :is ENTITY)
   (TURNAROUND-TIME)
   (MAXIMUM-TARDINESS)
   (AVERAGE-TARDINESS)
   (TOTAL-LATENESS)
   (AVERAGE-LATENESS)
   (NUMBER-OF-TARDY-BATCHES)
   (TOTAL-BATCHES-DONE))
```

BATCH Frame

Fig. 3. Examples of domain knowledge frames.

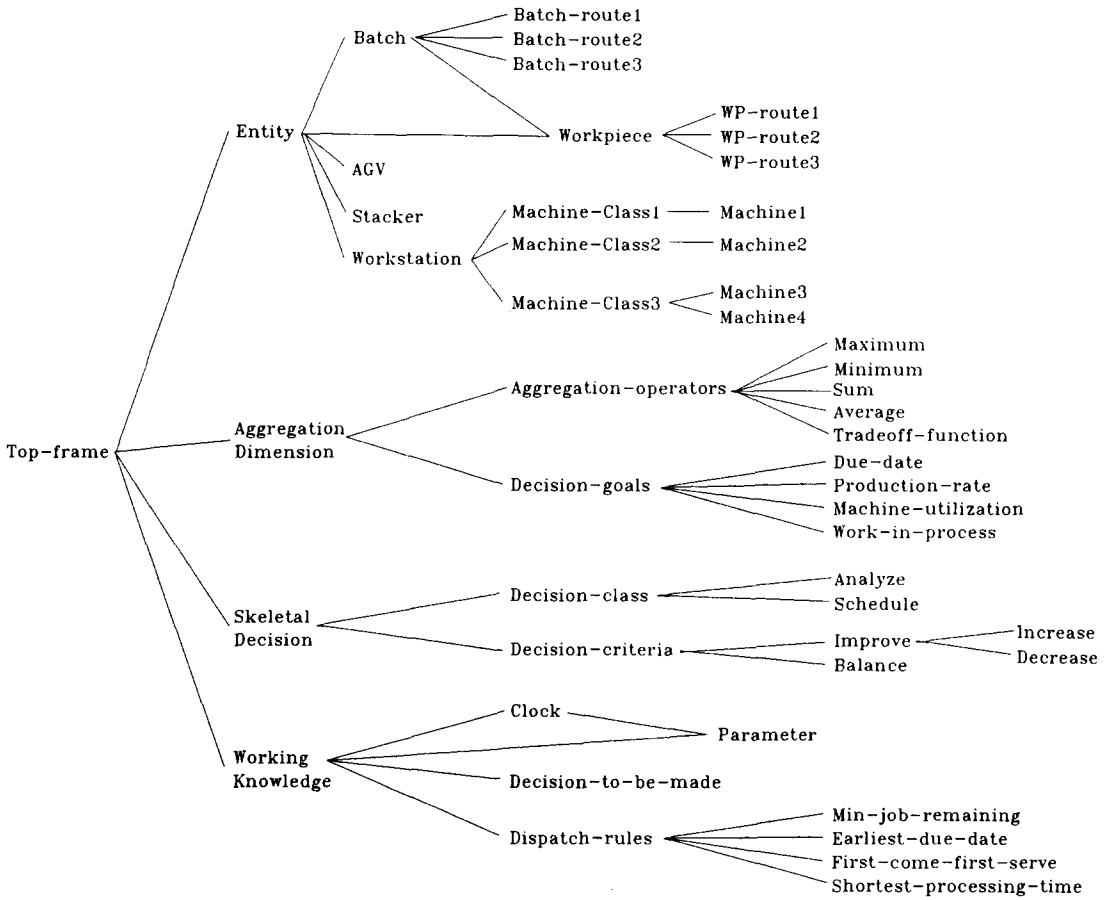


Fig. 4. Representation of domain knowledge.

4.3.1. Control module

The control module covers the major functioning of the FMS-DSS system. It is represented entirely as rules, partitioned into sets that relate to specific contexts. Rules are used to determine the most appropriate goal given the objectives specified by the user, choose the most appropriate aggregation strategy for the goal and context in question, as well as update the database if need be. Rules are also employed to determine if there is enough knowledge available to process this particular context. If not, the learning module is invoked to create the requisite knowledge. FMS-DSS's control module also employs a large set of working knowledge that relates to the context at hand. This working knowledge is implemented as frames, and is currently domain sensitive. More detailed descriptions of the system knowledge and structures are presented in [7]. The English-like version of a backward chaining control rule appears in fig. 5.

The antecedents of the backward rule represent the stages necessary for problem solving with FMS-DSS. Each stage comprises a forward chaining rule set, that performs a specific task. The first set classifies the objectives specified by the user in the concept-attribute framework, determines the goal concept to be created, creates the goal concept, and creates the context for the goal. The second rule set creates an instance of the

IF	The Aggregate-Concept is Created
AND	The Goal-Analysis is Finished
AND	Uncertainty is Resolved
AND	Classification is Complete
AND	Parameters are Filed
AND	All-Goals are Retracted
THEN	Mission is Accomplished

Fig. 5. Rule accomplish-mission.

relevant context of the goal concept, analyzes relevant goals, determines the aggregation operator required to meet the goals, analyzes the system's sample data, calculates the ratings for the goal attributes, and checks for noise and temporal uncertainty in the data. The third rule set resolves any uncertainty arising from noisy data by comparing the attribute values of the sample with the corresponding attribute-values of similar contexts in the database. The similarity between cases is determined by partial matching and selects the relevant parameters by classifying the historical data, and is described in more detail in the next subsection. Decision recommendations are made by the fourth rule set, wherein all classification tasks are completed. The fifth set of rules appends the sample data to the historical database, updates the context tree if an unusual situation occurred, instantiates the output frame, and writes the parameters to an output file. The rule set re-initializes the system for solving the next problem. The taxonomy of forward chaining rules used by FMS-DSS appears in fig. 6.

4.3.2. Noise reduction module

Excessive volume and noise in the data available to FMS-DSS is managed through abstraction. The abstraction techniques vary with the nature of uncertainty encountered. At present, the system is able to deal with temporal, contextual, and procedural uncertainty. Knowledge for handling uncertainty is also implemented as rules and aggregation hierarchies. The noise-reduction module, when activated by the control module, first determines the manifestation and the type of noise in the data, and then finds the remedy using a combination of clustering and case-based approach. This module can handle missing and contradictory values of a parameter or a goal attribute.

The contradiction-reduction algorithm analyzes the sample (status information from the simulation) for evidence of data inconsistency. The database is searched for an instance $?X$ based on the partial match on the parameters and the clock values. Goal attribute ratings are compared, and a difference of more than 20%

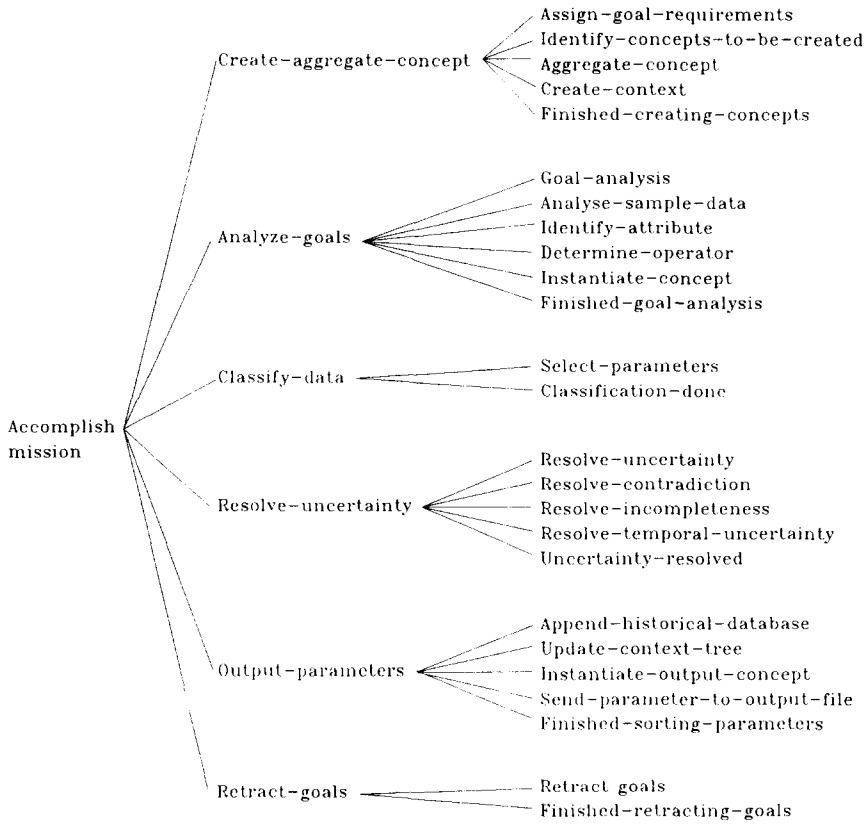


Fig. 6. Taxonomy of forward chaining rules.

suggests a contradiction may exist. The ratings on all relevant non-goal attributes are compared, and if they are within prescribed limits, it is inferred that the two scenarios are the same with contradictory goal attributes. The reasoning behind this is that if an abnormal situation occurred during the simulation run in question, it would affect non-goal attributes as well. Thus if the goal attribute was machine-utilization and if the machine did go down during the sampling period, its effect would be manifest in non-goal attributes such as WIP, AGV-utilization, production-rate, in addition to machine utilization. In cases of such inconsistency, the values of the goal attributes are discarded and replaced with reliable values from the instance *?X*. If the non-goal attributes also do not match, then it is inferred that the parameters are contradictory and the database is searched again, but this-time based on the partial match of clock and goal-attribute ratings. If the search is successful and the non-goal attributes match, then it is inferred that the two instances are the same and the parameters of the sample are corrected. If the find is unsuccessful or the non-goal attributes do not match, then it is inferred that the situation is *exceptional*. The contradiction reduction algorithm in fig. 7.

In this algorithm a variance of 10 and 20% is used to compare attribute ratings. This is the result of an experiment wherein ten identical simulations were run with different random number seeds. Allowing 10% in random variations in the performance ratings of simple, composite, and multiple goals covered almost all cases. Hence, a variance of 10% for single period comparisons, and a variance of 20% for two period comparisons was employed to allow for randomness of the system.

4.3.3. Learning module

The learning module is a LISP program based on conceptual clustering algorithm, INDUCE/2 [45] and includes aggregation operators minimum, maximum, variance, and fuzzy trade-off function, among others. If the available domain knowledge is insufficient for dealing with the current decision, the learning module tries to acquire new knowledge in an attempt to satisfy the goal. Functions performed by the learning module include selection of decision parameters for data generation via the simulation system, specification of the best and worst possible performance to date, computation of current performance relative to the extremes, identification of

```
Start
  Find an Instance ?X in the Database with same Parameters AND
     Clock values as the Instance ?Sample
     ;; The FMS is sampled at clock 5001, 10001, 15001 and
     ;; 20000
  Compare Goal-Attribute ratings
  If the variance in Goal-Attribute ratings is <= 20%
     Assert "Parameters OK" AND "Goal-Attributes OK"
     Exit
  Else Assert "Parameters may be Contradictory"
     Compare All Non-Goal-Attribute ratings
     If the variance in Non-Goal-Attribute ratings is <= 20%
        Assert "Parameters OK" AND "Goal-Attributes are Contradictory"
        Replace the ?Sample Goal-Attributes with ?X Goal-Attributes
        Assert Goal-Attributes corrected
        Retract ?Sample Uncertainty-Type
        Exit
     Else Find an Instance ?Y in the Database with same
           Clock value as ?Sample AND with variance in
           Goal-Attribute ratings <= 10%
        If the Find is successful AND
           Variance in Non-Goal-Attribute ratings is <= 10%
           Assert "Parameters are Contradictory"
           Replace ?Sample Parameters with ?Y Parameters
           Assert Parameters corrected
           Retract ?Sample Uncertainty-Type
           Exit
        Else Assert (Situation ?Sample Exceptional)
Exit
```

Fig. 7. Algorithm to analyze-resolve-contradiction.

IF There is an Instance *?Instance-Name* of Frame *?Context* with
Backlog-Limit as Unknown
Dispatch-Rule as Unknown
WP-Start-Rule as Unknown
Clock as *?clock*
AND There is no Instance of Frame Sampledata

THEN Send-Msg to *?Instance-Name* :Select-Best-Ratings
AND-THEN Display
FMS-DSS=>Parameters selected at clock = *?clock* are
(*?Blim ?Disrul ?Adrul*) and
Maximum achievable Performance-Rating is *?rating*

Fig. 8. Rule to select-decision-parameters.

decisions to be made for the current context, and update of the context tree to reflect new data collected by the system. Some of these functions are performed through production rules, while others employ LISP procedures. A sample rule that selects decision parameters appears in fig. 8, while the procedure for selecting the best cases is described in fig. 9.

Training examples needed for refinement of the domain knowledge are produced via an interface to an external knowledge augmenting system, which in this case is the simulation model. Some training examples may be deemed to be exceptional situations, i.e., when the sample data for a goal attribute falls outside the prior extremes for that attribute. In these cases the performance ratings need to be altered and the context tree updated. The parameters giving the highest performance rating may change due to the presence of an exceptional situation. As such, the context tree is updated continually to support dynamic decision making. This is done by invoking the method: *Update-context* which is presented in fig. 10. The method searches the context-tree for all the contexts affected by the exceptional situation, recalculates performance ratings, selects the best parameters, and updates the instances of the context. The functioning of the entire system as set of coherently interacting units is described in the next section.

5. FMS-DSS operation

Conceptually, FMS-DSS operates using an elegant three-stage procedure with provision for feedback in cases where performance is not deemed satisfactory, as depicted in fig. 11. The first stage comprises goal representation wherein user objectives are translated into a hierarchy of related goals using available domain-specific knowledge. In case the available knowledge is deficient, FMS-DSS employs a reasoning strategy that explicates the goal into known concepts. The second stage constitutes classification of current context vis-a-vis historical data. New concepts are learned or acquired through the use of a conceptual clustering technique, modified to provide a strong goal orientation. The final stage involves concept aggregation and evaluation. Semantic connections between the learned concepts and the highest-level goal are established at this stage. The effectiveness of the system is also tested at this stage through the use of an evaluation function. The use of evaluation functions allows for assessment of whether the goal has been satisfactorily met or whether further aggregation is warranted [13,36]. Goals that are partially met are further reclassified, whereas unmet goals mandate the creation of an alternative hierarchy from that currently employed.

At a procedural level, FMS-DSS functions in

Start
From All-Instances of the Frame Database
For the context list *?Goal-Context*
Calculate Total Performance-Rating by calling Handler :Calculate-Rating
Select the Instance *?X* with Maximum Performance-Ratings
Assert the Parameters and the Performance-Rating values in
the slots of Instance *?Instance-Name* of Frame *?Context*
Exit

Fig. 9. Algorithm to select-best-ratings.

```
Start
  Set all-slots of ?Sample to ?context-list
  Loop till ?context-list is exhausted
    Find the context
    Set all-instances to ?inst-list
    Loop till ?inst-list is exhausted
      If not updated
        Calculate Performance-Rating for the instance
          by calling the Handler :Calculate-Ratings
        Select best parameters by calling the Handler
          :Select-Best-Ratings
        Update the instance
        Flag the instance updated
Exit
```

Fig. 10. Algorithm to update-contexts.

the following manner, as characterized by the information flows in fig. 2. It does require an explicit means of generating an additional set of observations for learning. This is achieved through the use of an ECSL-based simulation of the FMS. The user, in this case an FMS scheduler, will make a scheduling or assignment request. The user is then queried for objectives to be achieved when meeting this request. Typical objectives seek to maximize workstation utilization, minimize lateness, and work-in-process. These objectives are reformulated as high-level goals to be met during this decision-making situation. The situation at hand is then abstracted into a context for manipulation. This process may require the aggregation and filtering of noisy data from the database. Contexts are then matched with decision rules for workpiece introduction and loading, based on knowledge culled from results of prior decisions. Simulation parameters specific to the context are generated. The final scheduling decisions, coupled with the newly generated simulation parameters, are then presented to the user, who may choose to override or modify them. The simulation system is then invoked using these parameters and decision rules. Information collected during this run is used to update the database. If the specific context produced results outside the performance limits experienced to date, then the domain knowledge is updated to reflect this new knowledge. If during the course of context creation or goal aggregation, the systems encounters a paucity of relevant domain knowledge, it invokes a learning module that utilizes prior data and new training examples generated from simulation runs to acquire the requisite knowledge, updating the domain knowledge and historical database in the process. Over time the system relies more on acquired knowledge than newly generated examples.

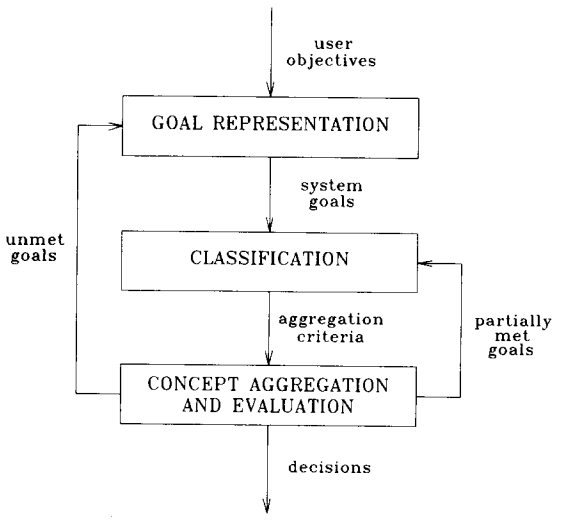


Fig. 11. Three stages of FMS-DSS operation.

The use of objectives specified by the user presents some difficulty in terms of evaluation of overall system performance. For example, an objective as maximizing workstation utilization, though constrained between 0 and 100%, is difficult to assess as a pure number. To counter this, FMS-DSS transforms the objective measure into a score based on a 0–1 scale, one representing the best performance achieved on this objective to date. The use of this scale provides an alternative frame of reference to assess performance during this decision making cycle. In case performance on this objective falls outside these limits, then the limits are re-adjusted, domain knowledge refined, and the database updated to reflect this additional information.

The use of multiple objectives confounds this approach in that objectives chosen by the user may be conflicting. For example, the maximization of workstation utilization and minimization of WIP are mutually conflict by nature. Clearly any chosen decision rule can at best perform well on one goal and poorly on the other, or average on both goals. Under these circumstances, the various decision rules available are evaluated on the 0–1 scale on each partial goal, and the user presented with the information. The user can then impose weights to choose among the alternatives presented. In addition to weights, the system can make a decision based on traditional

multiple attribute decision making strategies such as dominance, maximin, or conjunctive methods [19].

6. A sample session with FMS-DSS

This section presents a step by step discussion of an example scheduling session with FMS-DSS. It discusses the orders in which rules were fired, the demons or the methods used and for what purposes, and the assertions made.

Scheduling by FMS-DSS is a problem-solving process that requires six steps, three for decision-making, one for uncertainty management, and two for housekeeping and consistency maintenance. The user assigns a goal to FMS-DSS through a relation, GOAL-REQUIREMENT. A relation in GoldWorks is a special construct to structure assertions or set of assertions in the knowledge base. The goal assigned by the assertion, (GOAL-REQUIREMENT SCHEDULE IMPROVE DUE-DATE AVERAGE-TARDINESS BATCH), means that FMS-DSS should try to SCHEDULE BATCHES to IMPROVE AVERAGE-TARDINESS in order to meet DUE-DATES.

Upon receiving the goal from the user, FMS-DSS responds by issuing the query, (QUERY' (RUN-STATE ?*X* ACCOMPLISHED)), to the database and triggering the reasoning process. To match the pattern, run-state ?*X* accomplished, FMS-DSS starts by looking for assertions in the assertion base (or the working memory) that match the query pattern. If the search is unsuccessful, FMS-DSS next searches for the backward rule whose consequent matches the goal pattern. Here, the rule, ACCOMPLISH-MISSION'S consequent matches the goal pattern. The query can be satisfied only if all the six conditions represented by the antecedents of the backward rule are satisfied.

The first antecedent pattern "*run-state aggregate-concept created*", matches the enabling pattern of the rule-set, CREATE-AGGREGATE-CONCEPT. This rule-set is enabled, and the forward rules within this rule-set is now available to the system for forward chaining. The first rule, ASSIGN-GOAL-REQUIREMENTS, is put on the agenda because there are assertions to match its antecedents. The rule, ASSIGN-GOAL-REQUIREMENTS, fires making many assertions for the assertion base. Next, the rule, IDENTIFY-CONCEPT-TO-BE-CREATED, is put on the agenda as there are assertions to 'match its antecedents. Since no other rule can be put on the agenda at this time, this rule fires creating assertions (Concept-To-Be-Created #:SCHEDULE-IMPROVE-DUE-DATE) and

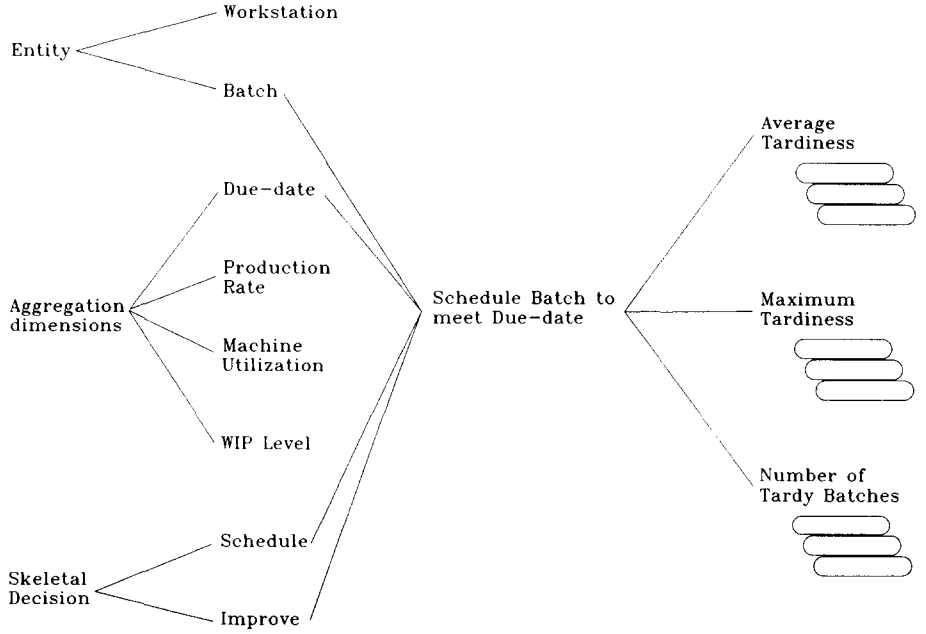


Fig. 12. Learning in FMS-DSS.

(Context-Name #:AVERAGE-TARDINESS) in the assertion base. Now, there are assertions to put rule, AGGREGATE-CONCEPT on the agenda. At this point, there is only one agenda item and the rule is fired. The firing of this rule creates a new frame #:SCHEDULE-IMPROVE-DUE-DATE and is put in its proper place in the taxonomy by making it a child-frame of each of the frames SCHEDULE, DUE-DATE, and BATCH (fig. 12). Next, the rule, CREATE-CONTEXT, is put on the agenda and fired creating the context frame, #:AVERAGE-TARDINESS, for the concept, #:SCHEDULE-IMPROVE-DUE-DATE. Finally, for the rule-set CREATE-AGGREGATE-CONCEPT, the rule, FINISHED-CREATING-CONCEPT is put on the agenda and fired. It asserts "run-state aggregate-concept created," and passes the control back to backward rule ACCOMPLISH-MISSION, thus satisfying the first condition.

The second rule-set, ANALYZE-GOAL, is now enabled. The rule, INSTANTIATE-GOAL-CONCEPT-AT-T0, fires to check if an instance of the active context already exists. If it does not, then the rule creates an instance of the active context frame. Next, rules are put on the agenda and fired according to their priorities to analyze the goal and to select the appropriate aggregation operators. The next rule to fire is SAMPLE-EXISTS, which checks to see if there is an instance of the frame, SAMPLE-DATA. Finally, the rule, FINISHED-GOAL-ANALYSIS, fires asserting "*run-state goal-analysis finished*" and passes the control back to the backward rule.

The third rule-set, RESOLVE-UNCERTAINTY, is activated next. It involves identifying noise in the data and resolving the resulting uncertainties which might be detrimental to the solution process. The available assertions in the working memory match the antecedent patterns of the rule, SAMPLE-WITH-TEMPORAL-UNCERTAINTY, and also the condition, rating at *T0* is greater than the rating at *T1*. In this situation, FMS-DSS immediately issues an alarm with the assertion, "*uncertainty-type is temporal-uncertainty*". The rule, ANALYZE-AND-RESOLVE-CONTRADICTORY-DATA, is now ready to fire. Firing this rule calls the method (a LISP routine), ANALYZE-RESOLVE-CONTRADICTION. This method first locates an instance $?X$ in the database with the same parameters and clock as the *?Sample* instance. It compares the goal attribute ratings of the two instances. Since the variance is greater than 10%, it next compares the non-goal attributes. It finds that the variance in the non-goal attributes of the two instances is $\leqslant 10\%$, which means the parameters are OK. The method then replaces the goal-attributes of *?sample* with the goal-attributes of $?X$, retracts *?sample* uncertainty-type.

The fourth step involves classification of data to satisfy the goal concept. When the rule-set, CLASSIFY-DATA, is enabled by the backward rule, the assertions in the assertion base match the antecedents of rules SELECT-PARAMETERS-AT-TO, and FINISHED-CLASSIFICATION. These rules are put on the agenda and fired in the order of their priorities. First the rule, SELECT-PARAMETERS-AT-T0 fires. It calls the method, SELECT-BEST-RATINGS, to select the parameters that will give the best performance rating. Finally, FINISHED-CLASSIFICATION, rule fires asserting "*run-state classification complete*". This completes scheduling as all the relevant parameters have been selected.

The fifth step involves writing the output parameters to a file which can be read by the simulation model. When the rule-set, OUTPUT-PARAMETERS, is enabled, rules are matched and fired in order of their priorities instantiating the output parameter frame and writing the parameters to an output file.

The sixth and last condition to meet the query requirement is accomplished by enabling the rule-set, RETRACT-ALL-GOALS. Two rules are put on the agenda and are fired in order of their priority. The first rule to fire calls the handler, RETRACT-ALL-SLOTS, to retract all assertions stored in the slots of the instance of the frame, DECISION-TO-BE-MADE. Finally, the rule, FINISHED-RETRACTING-GOALS, fires asserting "*run-state all-goals retracted*" and returns the control to the backward rule. With the last assertion, all five conditions of the backward rule are met and the query of the attempt is successfully completed with the assertion (RUN-STATE MISSION ACCOMPLISHED).

7. FMS-DSS evaluation

In absolute terms, it is difficult to demonstrate that FMS-DSS can effectively acquire knowledge for decision support in a real-time, complex envi-

ronment that may have large volume of data which may contain noise. In such an environment, the utility of the system can best be determined by comparison. One hundred and twenty-eight simulation runs were used to provide initial observations and to constitute the historical database. This database was used to acquire the knowledge about the FMS domain theory in a goal-directed manner. The acquired knowledge was later used in the experiments to make scheduling decisions.

Simple, composite, and multiple goals were used in the experiments. A simple goal involves using a single contextual attribute for the goal coming from a single concept, e.g., for the goal concept, MEET DUE-DATES, using minimize the number of tardy batches, involves using one attribute of a concept BATCHES as the contextual attribute. A composite goal addresses more than one contextual attribute for the goal concept coming from a single concept, e.g., for the same goal, MEET DUE-DATES, using minimize the number of tardy batches and also the average tardiness of a batch, involves using two attributes of the concept BATCHES as contextual attributes. A multiple goal inherits contextual attributes from two or more concepts, e.g., the goal, MEET DUE-DATES and IMPROVE MACHINE UTILIZATION, using maximize critical machine utilization and minimize the number of tardy batches, entails using attributes from two concepts BATCHES and WORKSTATIONS.

7.1. Empirical evaluation

To evaluate the performance of FMS-DSS a two-stage evaluation procedure was employed. In the first stage, the performance data from the simulation at period T_0 was routed through FMS-DSS. FMS-DSS evaluated the performance data of period T_0, and selected the necessary parameters to improve the performance of the system and communicated it to the simulation's decision module. This experiment was repeated for different goals. The performance of FMS-DSS was compared with that of different dispatching rules - earliest due-date (EDD), shortest processing time (SPT), minimum number of jobs remaining (MJR), and first come first serve (FCFS). The measure used for comparison was the rating on the goal attribute, 1 being the historical best and 0 being the historical worst. For composite and compound goals, ratings were cumulated. The results of average performance rating of this stage

Table 1
Comparison of FMS-DSS's performance

Goals ($n = 16$)	EDD	FCFS	MJR	SPT	FMS-DSS
Simple goals					
Maximum tardiness	0.90	0.64	0.48	0.58	0.93
No. of tardi bat	0.96	0.81	0.59	0.49	0.97
Work-in-process	0.54	0.55	0.60	0.46	0.84
Crit. mach. util	0.65	0.52	0.51	0.62	0.90
All mach. util	0.33	0.52	0.40	0.60	0.79
Stacker util.	0.44	0.53	0.53	0.66	0.94
AGV util.	0.45	0.52	0.55	0.65	0.92
WP output	0.48	0.48	0.54	0.72	0.91
Batch output	0.55	0.60	0.70	0.75	0.95
Composite goals					
Due-date (DD)	2.32	1.66	1.39	1.68	2.55
MHS	0.89	1.06	1.08	1.31	1.86
Production rate (PR)	1.86	2.08	2.19	2.36	3.06
Mach. util (MU)	0.98	1.05	0.91	1.21	1.47
Compound goals					
DD + MU	3.30	2.71	2.30	2.90	3.70
DD + WIP	2.77	2.11	1.79	2.22	3.04
DD + PR	4.18	3.74	3.58	4.04	5.08
DD + MU + WIP	3.76	3.16	2.70	3.45	4.20
DD + MU + WIP + PR	5.61	5.24	4.89	5.81	6.46

Table 2
Comparison of FMS-DSS's performance in the presence of noise

Goals ($n = 16$)	FMS-DSS w/o noise	FMS-DSS with noise
Simple goals		
Maximum tardiness	0.93	0.83
No. of tardi bat	0.97	0.81
Work-in-process	0.84	0.70
Crit. mach. util	0.90	0.74
All mach. util	0.79	0.68
Stacker util.	0.94	0.77
AGV util.	0.92	0.72
WP output	0.91	0.82
Batch output	0.95	0.78
Composite goals		
Due-date	2.55	2.27
MHS	1.86	1.53
Production rate	3.06	2.86
Mach. util	1.47	1.31
Compound goals		
DD + MU	3.70	3.57
DD + WIP	3.04	2.89
DD + PR	5.08	4.72
DD + MU + WIP	4.20	4.05
DD + MU + WIP + PR	6.46	6.20

is presented in table 1 indicating superior decision making through the use of FMS-DSS over traditional static dispatching rules.

During the second stage, experiments of stage 1 were repeated, except that data from the simulation was passed through a noise generator before it reached FMS-DSS, adding random noise to the data from simulation at period T_0. Types of noise included missing values for parameters and goal attributes and erroneous values for parameters and goal attributes. The performance of FMS-DSS with and without noise is presented in table 2.

As expected, the means of the performance rating for FMS-DSS without noise is greater than means of the performance rating for FMS-DSS with noise. However, the difference is not substantial. Even in the presence of noise, FMS-DSS outperformed the other scheduling heuristics, except for three goals – maximum tardiness, number of tardy batches, and due-date. All these occasions involve due-date considerations, and hence the earliest due-date rule did well. Further, it is evident that as more attributes are included in the goal, FMS-DSS clearly outperforms all the dispatching heuristics. Also, the difference between FMS-DSS with and without noise further diminishes due to balancing of the effects of noise on different attributes.

The reported tests do not completely reflect the power and scope of FMS-DSS. It employs a dynamic rating scheme, and the knowledge about prior decisions is stored in the context network. Instances of the context network consist of parameters determined by heuristic search of the highest rating. Scheduling knowledge contained in the context network is always kept current. If an exceptional situation occurs changing the historical maximum or the minimum, then the context network is updated based on the new ratings. Thus, the time spent on searching for best solution is reduced.

The true effect of dynamic rating scheme on decision is difficult to test. Although FMS-DSS allows the user to apply weights to the partial goals to indicate his/her preference of contextual attributes, in the experiments, for composite and multiple goals, cumulative partial goal ratings are used assuming that the user wants to put equal weights on all the specified contextual attributes. In order to make the ratings to have the same maximum and minimum base values for the attributes, ratings where calculated at the end of the experiments, i.e., experiments were based on static data.

7.2. *Learning in FMS-DSS*

FMS-DSS's learning process involves creating new descriptions of the goal concepts and their contexts, and placing them in the taxonomy of the domain-specific knowledge, depicted in fig. 12. Whenever a new goal is assigned to FMS-DSS, it first creates a hypothesis about description of the situation at hand. Next, it searches the database for instances in support of its hypothesis. Depending upon the strength of the match, it creates new concepts or instantiates a context of an existing concept, thus adding new knowledge in the knowledge base as a generalization of the encountered situation. FMS-DSS will always find a match for a goal concept, but the degree of match may vary. Thus, if FMS-DSS encounters a totally new situation, it will still help the user in finding the instances that describe the goal. Initially, however, the confidence in the knowledge

will be low. As more instances become available, the knowledge base is "fine-tuned", and the confidence increases.

Although the learning in FMS-DSS involves acquiring new concepts, it can also be seen as acquiring new rules. For instance, the knowledge acquired by solving the problem in the above session can be represented by the following rule.

IF *The scheduling goal is to meet the due-data at period*
AND *The context is number-tardy-batches*
THEN *Select parameters (X Y Z)*

This is clearly a learning episode as it recognizes and represents the relationship between the scheduling goal and the decision parameters under the system conditions described by the attributes of the instances of the context, and can be used for future decision making without going through the aggregation process. The benefits realized through the use of FMS-DSS as a scheduling system make this approach attractive as these benefits are unique to this system. They include:

Complexity reduction. Complexity is reduced by grouping several details concerning entities into a single concept in an intelligent manner. This allows for reduced control effort and more efficient utilization of control resources.

Object identification. FMS-DSS can recognize objects as a member of familiar concepts instead of unique occurrences. Correspondingly, it is able to generalize its learning experiences to larger instances of observations.

Redundancy reduction. Once FMS-DSS recognizes a concept, it applies previous experience instead of relearning it, a decided advantage over systems wherein the entire decision process needs to be performed for every instance.

Uncertainty reduction. Through its ability to classify instances with missing, incorrect, or otherwise inaccurate information, FMS-DSS can eliminate some of the uncertainty involved in the case of incomplete information situations.

System's action direction. FMS-DSS directs appropriate action to be taken by simulation after having classified the object.

Events relation. FMS-DSS classifies the events and attributes and establishes the causal and temporal relationships between them, thus making it easier to understand working relationships in the dynamic system. FMS-DSS is not without its disadvantages. As in the case of any inductive learning system, FMS-DSS may be prone to improper generalizations in the face extremely imprecise situations. The knowledge acquired by the system is dependent on the set of training data employed–clearly, poor examples will engender substandard domain knowledge. When presented with a situation with radically different assumptions than those encountered before, FMS-DSS's performance may be questionable. Lastly, in its current implementation in a LISP environment, FMS-DSS is a resource-intensive system.

8. Conclusions

This paper enumerates some problems with current approaches to decision making for real-time systems and discusses the need for new methods. It then presents FMS-DSS, a decision support system based on goal-directed conceptual aggregation for supporting FMS scheduling and control decisions. Conceptual aggregation is an AI approach which applies a modified conceptual clustering technique and case-based learning for explaining and summarizing data. It also has the capability of dealing with the imperfections of data such as inconsistency, contradiction, and incompleteness. Results show that upon integration with machine learning, DSS can become a powerful tool for supporting, and perhaps making, real-time decisions.

References

[1] R.H. Bonczek, C.W. Holsapple and A.B. Whinston, Future Directions for Developing Decision Support Systems, Decision Sciences 11, (1980) 616–631.

[2] L.B. Booker, D.E. Goldberg and J.H. Holland, Classifier Systems and Genetic Algorithms, Artificial Intelligence 40, No. 1–3 (1989) 235–282.

[3] W. Buntine and D. Stirling, Interactive Induction, Proceedings of the Fourth International Conference on Artificial Intelligence Applications, San Diego, CA (1988) 320–326.

[4] J.G. Carbonell, Learning by Analogy: Formulating and Generalizing Plans from Past Experience, in: R.S. Michalski, J.G. Carbonell and T.M. Mitchell, Eds., Machine Learning: An Artificial Intelligence Approach (Tioga Press, Palo Alto, CA 1983) 137–161.

[5] A.R. Chaturvedi, Goal-Directed Conceptual Aggregation: An Artificial Intelligence Techniques to Support Management Decision Making, Doctoral Dissertation, University of Wisconsin, Milwaukee (1989).

[6] A.R. Chaturvedi and G.K. Hutchinson, A Model for Simulating AGV Congestion in An FMS, Paper No. 975, Krannert Graduate School of Management, Purdue University, West Lafayette, IN (1990).

[7] A.R. Chaturvedi, G.K. Hutchinson and D.L. Nazareth, FMS Scheduling Using Goal-Directed Conceptual Aggregation, Proceedings of the Seventh IEEE Conference on Artificial Intelligence Applications, Miami Beach, FL (1991) 315–321.

[8] K. Chen and C.-Y. Lu, A Machine Learning Approach to the Automatic Synthesis of Mechanistic Knowledge for Engineering Decision Making Proceedings of the Fourth International Conference on Artificial Intelligence Applications, San Diego, CA (1988) 306–311.

[9] R. Davis and D.B. Lenat, Knowledge-based Systems in Artificial Intelligence, (McGraw Hill Book Co., New York, 1982).

[10] G. DeJong and R. Mooney, Explanation-Based Learning: An Alternative View, Machine Learning 1, No. 2 (1986) 145–176.

[11] H.B. Eom and S.M. Lee, A Survey of Decision Support System Applications (1971–April 1988), Interfaces 20,No. 3 (1990) 65–79.

[12] R.E. Fikes, P.E. Hart and N.J. Nilsson, Learning and Executing Generalized Robot Plans, Artificial Intelligence 3, No 4 (1972) 251–288.

[13] D. Fisher, Improving Inference Through Conceptual Clustering, Proceedings of the Sixth National Conference on AI, Seattle, WA (1987).

[14] G.A. Gorry and M.S. Scott Morton, A Framework for Management Information Systems, Sloan Management Review 13, No. 1 (1971) 55–70.

[15] R.P. Hall, Computational Approaches to Analogical Reasoning: A Comparative Analysis, Artificial Intelligence 39, No. 1 (1989) 39–120.

[16] G.E. Hinton, Learning in Parallel Networks, BYTE Magazine 10 (1985) 4.

[17] G.E. Hinton, Connectionist Learning Procedures, Artificial Intelligence 40, No. 1–3 (1989) 185–234.

[18] J.H. Holland, Adaptation in Natural and Artificial Systems (University of Michigan Press, Ann Arbor, MI, 1975).

[19] C.-L. Hwang and K. Yoon, Multiple Attribute Decision Making – Methods and Applications: A State-of-the-Art Survey, (Springer-Verlag, Berlin, 1981).

[20] S.T. Kedar-Cabelli, Analogy: A Unified Perspective, in: D.H. Helman, Ed., Analogical Reasoning (Reidel, Dordrecht, Netherlands 1988).

[21] P.G.W. Keen and M.S. Scott Morton, Decision Support Systems: An Organizational Perspective (Addison Wesley, Reading, MA, 1978).

[22] D.B. Lenat, The Nature of Heuristics, Artificial Intelligence 19, No. 2 (1982) 189–249.

[23] D.B. Lenat, Theory Formation by Heuristic Search. The Nature of Heuristics II: Background and Examples, Artificial Intelligence 21, No. 1,2 (1983) 31–59.

[24] D.B. Lenat, EURISKO: A Program that Learns New Heuristics and Domain Concepts. The Nature of Heuristics III: Program Design and Results, Artificial Intelligence 21, No. 1,2 (1983) 61–98.

[25] D.B. Lenat and J.S. Brown, Why AM and EURISKO Appear to Work, Artificial Intelligence 23, No. 3 (1984) 269–294.

[26] T.P. Liang, Modeling by Analogy: A Case-based Approach to Automated Linear Program Formulation, Proceedings of the 24th Annual Hawaii International Conference on System Sciences, Vol 4. (1991) 276–283.

[27] T.P. Liang and B.R. Konsynski, Modeling by Analogy: Use of Analogical Reasoning in Model Management Systems, Proceedings of the 1990 ISDSS Conference (1990) 405–421.

[28] L.T. McCarty and N.S. Sridharan, The Representation of an Evolving System of Legal Concepts II: Prototypes and Definitions, Proceedings of the Seventh International Joint Conference on Artificial Intelligence, Vancouver, BC, Canada (1981) 246–253.

[29] R.S. Michalski, Discovering Classification Rules using Variable-Valued Logic System VL1, Proceedings of the Third International Joint Conference on Artificial Intelligence (1973) 162–172.

[30] R.S. Michalski and R. Chilausky, Knowledge Acquisition by Encoding Expert Rules Versus Computer Induction from Examples: A Case Study Using Soybean Pathology, International Journal of Man-Machine Studies 12, No. 1 (1980) 63–87.

[31] R.S. Michalski and R.E. Stepp, Learning from Observation: Conceptual Clustering, in: R.S. Michalski, J.G. Carbonell and T.M. Mitchell, Eds., Machine Learning: An Artificial Intelligence Approach (Tioga Publishing Company, Palo Alto, CA, 1983) 331–363.

[32] S. Minton, J.G. Carbonell, C.A. Knoblock, D.R. Kuokka, O. Etzoni and Y. Gil, Explanation-Based Learning: A Problem Solving Perspective, Artificial Intelligence 40, No. 1–3 (1989) 63–118.

[33] T.M. Mitchell, R.M. Keller and S.T. Kedar-Cabelli, Explanation-Based Generalization: A Unifying View, Machine Learning 1, No. 1 (1986) 47–80.

[34] S.C. Park, N. Raman and M.J. Shaw, Heuristic Learning for Pattern Directed Scheduling in a Flexible Manufacturing System, Proceedings of the Third ORSA/TIMS Conference on Flexible Manufacturing Systems (Elsevier, Amsterdam 1989) 369–376.

[35] J.R. Quinlan, Semi-Autonomous Acquisition of Pattern-based Knowledge, in: D. Michie, Ed., Introductory Readings in Expert Systems (Gordon and Breach Science Publishers, New York, 1982) 192–207.

[36] L. Rendell, Substantial Constructive Induction using Layered Information Compression: Tractable Feature Formation in Search, Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Los Angeles, CA (1985) 650–658.

[37] F. Rosenblatt, The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain, Psychological Review 65 (1958) 386–407.

[38] F. Rosenblatt, Principles of Neurodynamics and the Theory of Brain Mechanisms (Spartan Books, Washington, DC, 1962).

[39] A.L. Samuel, Some Studies in Machine Learning Using

the Game of Checkers, IBM Journal of Research and Development 3, No. 3 (1959) 210–229.

[40] A.L. Samuel, Some Studies in Machine Learning Using the Game of Checkers. II – Recent Progress, IBM Journal of Research and Development 11, No. 6 (1967) 601–617.

[41] M.J. Shaw, Applying Inductive Learning to Enhance Knowledge-based Expert Systems, Decision Support Systems 3, No. 4 (1987) 319–332.

[42] M.J. Shaw, Knowledge-based Scheduling in Flexible Manufacturing Systems: An Integration of Pattern-Directed Inference and Heuristic Search, International Journal of Production Research 26, No. 5 (1988) 821–844.

[43] M.J. Shaw and A.B. Whinston, An Artificial Intelligence Approach to the Scheduling of Flexible Manufacturing Systems, IIE Transactions 21, No. 2 (1989) 170–183.

[44] R.H. Sprague and E.D. Carlson, Building Effective Decision Support Systems (Prentice Hall, Englewood Cliffs, NJ, 1982).

[45] R.E. Stepp and R.S. Michalski, Conceptual Clustering of Structured Objects: A Goal-Oriented Approach, Artificial Intelligence 28, No. 1 (1986) 43–69.

[46] K.Y. Tam, Automated Construction of Knowledge-Bases from Examples, Information Systems Research 1, No. 2 (1990) 144–167.

[47] P.H. Winston, Learning Structural Descriptions from Examples, in: P.H. Winston, Ed., The Psychology of Computer Vision (McGraw Hill, New York, 1975).

[48] P.H. Winston, Learning and Reasoning by Analogy, Communications of the ACM 23, No. 12 (1980) 689–703.