1996

Development of a Predictive-Reqactive Scheduler Using Genetic Algorithms and Simulation-based Scheduling Software

Albert Jones
National Institute of Standards and Technology
Metrology Bldg Room A127
Gaithersburg, MD 20899-001
USA
301-975-3554
jonesa@cme.nist.gov

Frank Riddick
National Institute of Standards and Technology
Metrology Bldg Room A127
Gaithersburg, MD 20899-001
USA
301-975-3892
riddick@cme.nist.gov

Luis Rabelo
Ohio University
Stocker Engineering Center
Athens, OH 45701-2979
USA
614-593-1542
rabelo@bobcat.ent.ohiou.edu

SYNOPSIS

Simulation-based scheduling packages are widely used in manufacturing plants around the world. These packages include a large number of "canned" dispatching rules which produce schedules for both simple and complex production environments. Users can extend these canned rules by adding their own plant-specific rules. In this paper we describe an iterative approach in which we first generate schedules using an optimal search technique, genetic algorithms (GA), and then predict how well those schedules perform using the simulation-based scheduling software. We also describe an information model for a status database which provides the basis for using this iterative approach for reactive scheduling as well.

INTRODUCTION

Discrete-event, simulation-based scheduling packages are widely used in manufacturing plants around the world. These packages include a large number of "canned" dispatching rules which will produce schedules for both simple and complex production environments. Users can extend these canned rules by adding their own plant-specific rules. It is important to note that these rules are guaranteed to produce only feasible schedules. The notion of optimality (or near-optimality) with respect to one or more performance measures is typically not considered. In fact, the knowledge about which rule(s) to use to achieve some desired performance measure(s) must be supplied by the user or derived from extensive experimentation with the packages.

In this paper we describe a new iterative approach in which we first generate schedules using a genetic algorithm (GA). The GA is an optimal search technique which generates candidate schedules. Candidate schedules are evaluated using a commercial simulation-based scheduling package. The evaluation will predict how well the system will perform if the resulting schedule is implemented. This "generate-evaluate" loop continues until the optimal solution has been found, or some threshold limit has been reached. By optimality we mean that no further improvement in the performance measure(s) can be found. Hence, this approach will lead to the best schedule for the given performance measure(s). By threshold limit we mean a search time (like 1 second) limit or a maximum number of candidates (like 100 schedules) to evaluate. Because the GA is very fast, and the simulations are completely deterministic, this approach can be used to modify existing schedules (reactive scheduling) based on feedback from the shop floor.

The paper is organized as follows. In section one, we summarize the technique which generates schedules based on genetic algorithms. In section two, we provide the EXPRESS information model for the status database which is used to do reactive scheduling. In the final section, we describe out future plans.

SCHEDULING USING GENETIC ALGORITHMS

We now describe a methodology for solving the real-time sequencing and scheduling problems using genetic algorithms.

Genetic Algorithms (GA)

Genetic algorithms (GA) are an optimal search technique based on a direct analogy to Darwinian natural selection and mutations in biological reproduction. In principle, genetic algorithms encode a parallel search through concept space, with each process attempting hill climbing [15]. Instances of a concept correspond to individuals of a species. Induced changes and recombinations of these concepts are tested against an evaluation function to see which ones will survive to the next generation.

The use of genetic algorithms requires five components:

1. A way of encoding solutions to the problem - fixed length string of symbols.
2. An evaluation function that returns a rating for each solution.
3. A way of initializing the population of solutions.
4. Operators that may be applied to parents when they reproduce to alter their genetic composition such as crossover (i.e. exchanging a randomly selected segment between parents), mutation (i.e. gene modification), and other domain specific operators.
5. Parameter setting for the algorithm, the operators, and so forth.

The following example illustrates the application of both the blind recombination and partially mapped crossover (PMX) operators [8] to a simple sequencing problem with 7 job types. Each job-type has its own arrival time, due date, and processing time distributions. The objective is to determine a sequence that minimizes Maximum Tardiness.

Table 1. The sequencing problem

Queue Pos.	Job Type	Arrival Time	Processing Time	Due Date
1	6	789	Normal(8,0.4)	890
2	6	805	Normal(8,0.4)	911
3	5	809	Normal(10,0.6)	910
4	1	826	Normal(4,0.2)	886
5	2	830	Normal(6,0.3)	905
6	7	832	Normal(15,0.75)	1009
7	6	847	Normal(8,0.4)	956
8	3	848	Normal(5,0.2)	919
9	1	855	Normal(4,0.2)	919
10	4	860	Normal(3,0.1)	920

The set-up time for these jobs is sequence dependent as shown in Table 2.

Table 2. Set-Up Times Dependencies

		Previous Job Type						
		1	2	3	4	5	6	7
	1	0	1	2	2	3	2	2
	2	1	0	2	3	4	3	2
Current	3	2	2	0	3	4	3	2
Job Type	4	1	2	2	0	4	4	2
	5	1	2	2	3	0	3	2
	6	1	2	2	3	3	0	3
	7	1	2	2	2	2	2	0

The simple procedure described below is used to resequence the 10 jobs in the queue until no improvement in the objective function can be found (i.e. we have the optimal solution).

1. Randomly generate n feasible sequences e.g., n=50.
2. *Compute the tardiness for each sequence and rank from lowest to highest.*
3. Choose the m best sequences, m < n, e.g. m=25.
4. Use blind recombination operator to produce a new set of n feasible sequences.
5. Randomly select pairs of sequences. Apply PMX operator.
6. Randomly select a pair of jobs in each offspring and mutate (switch) them.

The PMX operator generates offspring by randomly selecting a swapping interval between two crossover points and switching the jobs. These offspring will inherit the elements from the interval of one of the parents. If the resulting sequence is infeasible (has one or more duplicate jobs) the infeasibility must be removed through a technique known as mapping and exchanging. For example, consider two sequences (A and B):

Position:	1	2	3	4	5	6	7	8	9	10	
A(Job Numbers)		9	8	4	5	6	7	1	3	2	10
B(Job Numbers)		8	7	1	2	3	10	9	5	4	6

If the swapping interval 4 to 6 were selected, then the application of PMX would yield

Position:	1	2	3	4	5	6	7	8	9	10	
A'		9	8	4	2	3	10	1	3	2	10
B'		8	7	1	5	6	7	9	5	4	6

two infeasible sequences. Now, we apply the mapping and exchanging technique to remove duplicate jobs. This yields the following two feasible offsprings.

Position:	1	2	3	4	5	6	7	8	9	10	
A"		9	8	4	2	3	10	1	6	5	7
B"		8	10	1	5	6	7	9	2	4	3

These offspring do not necessarily replace their parents in the population; they are simply placed among the rest of the potential solutions according to their ranking.

A number of approaches have been utilized in the application of genetic algorithms (GA) to job shop scheduling problems [8, 16]. (references). Most relevant to this research is the work presented in Starkweather et al. [26]. They were the first to use genetic algorithms and on-line simulation to solve a dual-criteria job shop scheduling problem in a real production facility. Those criteria were the minimization of average inventory in the plant and the minimization of the average waiting time for an order to be selected. Their approach generated schedules which produced inventory levels and waiting times which were acceptable to the plant manager. In addition, the integration of the genetic algorithm with the on-line simulation made it possible to react to changing system dynamics.

Selecting initial schedules

The key to using genetic algorithms effectively in solving scheduling problems lies in the generation of "good" initial schedules. In this research, we are using neural networks to select dispatching rules which are then used to generate these initial schedules. This approach was used because 1) neural networks are very fast and learn simple relationhsips quickly, and 2) the performance of the selected dispatching rules on real systems could be easily evaulated by simulation-based scheduling software. Specifically, as indicated in Figure 1, we use a backpropagation neural network to rank the available rules for **each** performance measure of interest (This allows us to handle single or multiple performance measures.)

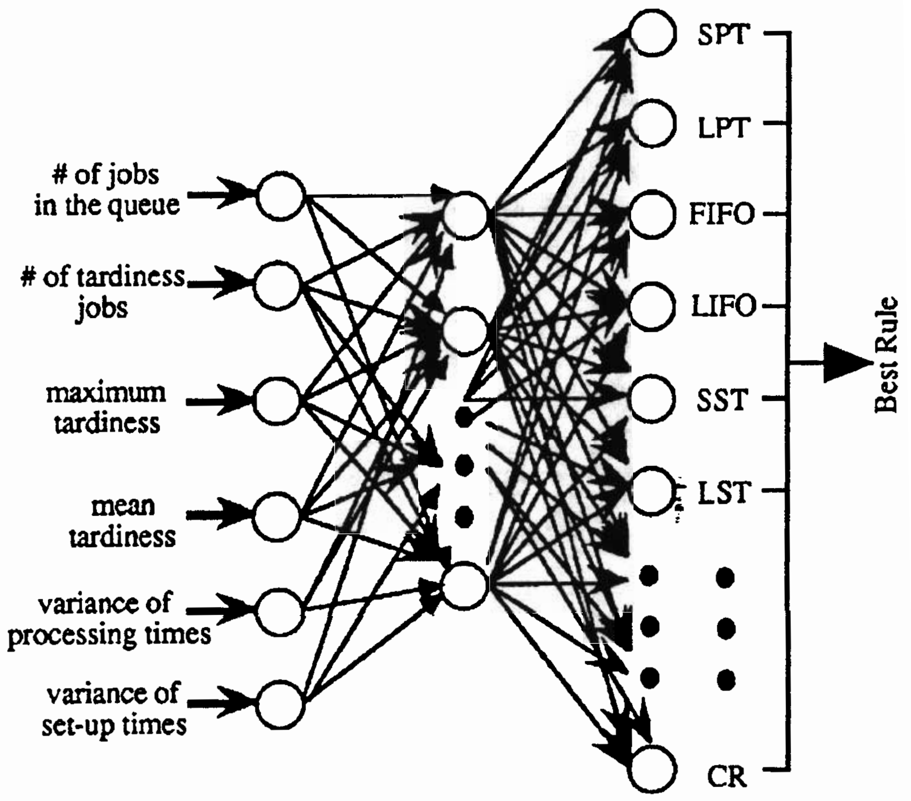

Fig 1 Neural Network for Ranking Dispatching Rules

The weights for each of the nodes in these networks are selected after a thorough training. To carry out this training, we first generate training data sets from a large collection of simulation studies.

Suppose we wanted to train a neural net to minimize the maximum tardiness and we wanted to consider the following dispatching rules: SPT, LPT, FIFO, LIFO, SST, LST, CR, etc. (See Figure 1). After simulating each of these rules off-line under a variety of input conditions, we would be able to rank them to determine the rule which minimizes the maximum tardiness under different conditions. We would then use these results to train (i.e., choose weights) a neural net. Weight selection is done using the gradient-descent technique in a feed-forward network. The training attempts to minimize the cost function: $C(W) = ½\Sigma\ (T_{ip} - O_{ip})$ where the T_{ip} is the target value, O_{ip} is the output of network, i is the output nodes, and p is the training patterns. After the network propagates the input values to the output layer, the error between the desired output and actual output will be "back-propagated" to the previous layer. In the hidden layers, the error for each node is computed by the weighted sum of errors in the next layer's nodes. In a three-layered network (see Figure 2), the next layer means the output layer. If the activation function is sigmoid, the weights are modified according to

$$\Delta W_{ij} = h\, X_j\, (1 - X_j)(T_j - X_j)\, X_i \qquad (1)$$

or

$$\Delta W_{ij} = h\, X_j\, (1 - X_j)\, (\Sigma\, d_k W_{jk})\, X_i \qquad (2)$$

where W_{ij} is weight from node i to node j, h is the learning rate, X_j is the output of node j, T_j is the target value of node j, d_k is the error function of node k.

If j is in the output layer, relation (1) is used. Relation (2) is for the nodes in the hidden layers. The weights are updated until the cost function is minimized.

An example

Using the data from Tables 1 and 2, we first ran the neural network for min Maximum Tardiness for thirteen dispatching rules (see Table 3). The ranking from the neural network was SPT, SPST, EDD, mSLACK, and CR. The actual results from a simulation we ran are shown in the Table 3. From the table, we can see that there was a very high correlation between the rankings from the nerual network and the actual simulation results.

We then used the SPT and mSLACK sequences {6 8 9 10 7 3 4 1 2 5}, {6 3 8 1 4 7 9 10 2 5} as inputs to our Genetic Algorithm. We stopped the search after 200 ms, and examined the results. The sequence was {4 3 6 9 10 8 7 1 2 5} which produced a Maximum Tardiness of 4. In less than 1 second, the optimal sequence was found to be {8 5 4 1 2 3 9 10 6 7} with a Maximum Tardiness of 2. It is interesting to note, that this sequence could not be generated using any of the dispatching rules we tried. Because of this, we have initiated a research effort to use inductive learning to derive rules frmothis schedule which could eventually be put into some rule base for future use.

The main conclusion to be drawn from these results is using a commercial scheduling package based strictly on dispatching rules, can lead to solutions which are far from optimal. To overcome this problem we are attempting to integrate this schedule generation technique based on genetic algorithms with two commercial scheduling packages. We will update our progress on this effort at the conference.

Table 3. Neural Network Results

Rule	Rank	Max Tardiness	Job Sequence

SPT	1	7	6 8 9 10 7 3 4 1 2 5
LPT	8	57	1 2 5 4 3 7 6 8 9 10
FIFO	7	54	1 2 3 4 5 6 7 8 9 10
LIFO	5	42	10 9 8 7 6 5 4 3 2 1
SST	7	54	4 1 2 5 3 7 6 8 9 10
LST	6	43	3 1 4 6 2 7 8 5 9 10
SPST	1	7	6 8 9 10 7 3 4 1 2 5
LSPT	8	57	1 2 5 4 3 7 6 8 9 10
EDD	3	12	6 8 3 7 1 9 4 10 2 5
LDD	9	74	5 2 10 4 1 9 7 3 8 6
mSLACK	2	11	6 3 8 1 4 7 9 10 2 5
CR	3	12	1 6 3 8 4 7 9 10 2 5
sSLACK	4	18	8 10 6 9 7 3 4 1 5 2

PREDICTIVE and REACTIVE SCHEDULING

Whenever a new job enters the system, we want to generate a schedule which will predict when that job will finish, and its impact on a variety of system performance measures. Whenever conditions change on the shop floor, we want to be able to react to these changes by modifying the current schedule. To do this, we must know the shop floor "status". This status is used as input to the neural networks and the simulations. We have been working with vendors and users to define this concept of "status". The following preliminary data model, written using the express data modelling language, is used to capture the relevant information units for maintaining status.

```
SCHEMA shop_floor_status ;

TYPE id = STRING ;
END_TYPE ;
(* Generic id type                                                    *)

TYPE real_quantity = REAL;
END_TYPE;
(* Specifies a quantity of something which  may have fractional parts *)

TYPE whole_quantity = INTEGER ;
END_TYPE;
(* Specifies a whole quantity of something                            *)

TYPE percentage = REAL;
END_TYPE;
(* Specifies a percentage of something                                *)

TYPE elapsed_time = REAL;
END_TYPE ;
(*  Specifies the seconds since an event                              *)
```

```
TYPE load_status  =   ENUMERATION OF
   (started, not_started, ended, interrupted, restarted, held, released);
END_TYPE;
(* Specifies the processing state of a load.
 * started -----> load is processing normally
 * not_started -> load has never been in started state
 * ended -------> all processing for the load has completed
 * interrupted -> a previously started load's processing has been stopped for unknown reasons
 * restarted ---> a previously interrupted or held load is now processing normally
 * held --------> a previously started load's processing has been manually stopped
 * released ----> a previously held load is ready to be restarted
 *)

TYPE resource_type  =  ENUMERATION OF
  (operator, machine, tool, fixture);
END_TYPE ;
(* Specifies the kinds of resources in a factory.
 * operator ----> a human resource
 * machine  ----> a device which uses tools to manufacture parts
 * tool --------> a implement used by a machine or operator to modify parts
 * fixture -----> a device which hold parts to facilitate their
 *              modification by other resources
 *)

TYPE resource_status =
  ENUMERATION OF
  (available, busy, setup, tear_down, pm, broken, blocked, off_shift);
END_TYPE ;
(* Specifies the processing state of a resource.
 * available ---> resource is not being used
 * busy --------> resource is processing a load
 * setup -------> resource is involved in a setup process for itself or another resource
 * tear_down-> resource is involved in a tear down process for itself or another resource
 * pm ---------> resource is involved in preventative maintenance
 * broken   ---> resource is broken
 * blocked ---> some external force is preventing the resource from transitioning to its next
 *                 state
 * off_shift ---> resource is currently not busy and not available
 *)

ENTITY timestamp ABSTRACT SUPERTYPE;
  year, month, day,
  hour, minute, seconds: NUMBER ;
END_ENTITY ;
(* Concrete types based on this abstract type will specify
 * attributes for year, month, day, hour, minutes and second.
 *)
```

```
ENTITY product ;
 id:              id ;                    -- Identifier for a product
END_ENTITY ;
(* Defines the state for a part or sub-assembly that is being produced
 * in the factory.
 * Only the id attribute is needed for maintaining factory status.
 *)

ENTITY order ;
 id:              id ;                    -- Identifier for an order
END_ENTITY ;
(* Defines the state for a request to build a particular product as part
 * of a load.
 * Only the id attribute is needed for maintaining factory status.
 *)

ENTITY jobstep;
 id:              id ;                    -- Identifier for a jobstep
END_ENTITY ;
(* Defines the state for a manufacturing operation that will be
 * performed on a load.
 * Only the id attribute is needed for maintaining factory status.
 *)

ENTITY load ;
 id:                    id ;             -- Identifier for a load
 product:               product ;        -- The product being produced
 order:                 order ;          -- The order which created the load
 current_amount:        whole_quantity ;-- Current # of parts in a load
 start_amount:          whole_quantity ;-- Beginning # of parts in a load
 start_date:            timestamp ;     -- The planned starting date
 due_date:              timestamp ;     -- The planned completion date
 release_date:          OPTIONAL timestamp ;  -- The actual start date
 resources:             SET OF resource ;  -- The resources currently associated with this load
 current_jobstep:       jobstep ;        -- The currently associated jobstep
 current_jobstep_start_time: timestamp ;       -- The time when this started
 pieces_complete_percentage: percentage ;  -- The percent of the load which is complete
 up_time_logged:        elapsed_time ; -- The amount of time this load has actually been
                                        -- processing
 previous_state:        load_status;     -- The previous processing state of the load
 current_state:         load_status;     -- The current processing state of the load
UNIQUE
 id;                                     -- Specifies the id uniquely identifies a load
INVERSE
 associated_buffer: OPTIONAL buffer FOR contents ; -- The buffer which contains the load
END_ENTITY ;
(* Defines the state for collection of products upon which manufacturing operations will be
 * performed.
 *)

ENTITY resource;
  id:                     id ;              -- Identifier for a resource
  type:                   resource_type ; -- The resources type
  previous_state:         resource_status ; -- The previous status of the resource
  current_state:          resource_status ; -- The current status of the resource
  last_product_processed: product ;  -- The product last associated with the resource
  resource_usage:         elapsed_time ; -- The total amount of time a resourcehas been used
                                     -- since it has been into service or refreshed
  time_of_last_update: timestamp ;  -- The time of the last update
UNIQUE
  id;                                -- Specifies the id uniquely identifies a resource
INVERSE
  current_load:           OPTIONAL load FOR resources;
                                     -- The load associated with this resource
END_ENTITY;
(* Defines the state for things which will be used to manufactureproducts.  Resources may be
 * operators, machines, tools, or fixtures.
 *)


ENTITY buffer;
  id:                     id ;              -- Identifier for a buffer
  contents:               LIST OF load; -- The loads contained in the buffer
UNIQUE
  id;                                -- Specifies the id uniquely identifies a buffer
END_ENTITY ;
(* Defines the state of an area used to temporarily hold loads.            *)

ENTITY material;
  id :                    id ;              -- Identifier for a material
  level:                  real_quantity;  -- Specifies how much material there is
UNIQUE
  id;                                -- Specifies the id uniquely identifies a material
END_ENTITY ;
(* Defines the level of a consumable item that is used in manufacturing processes.
 *)
END_SCHEMA;
```

We are planning to use these models as the basis for integrating commercial shop floor data collection systems, the genetic algorithms, and commercial simulation-based scheduling packages. Eventually, we hope to have the genetic algorithm as part of the scheduler.

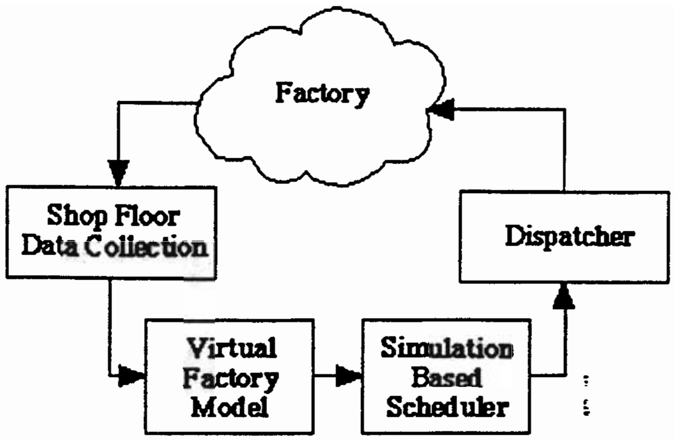

Fig 2 Overview of Integrated System

FUTURE WORK

In this paper, we have described a hybrid scheduling approach which combines neural networks and genetic algorithms. Neural networks select dispatching rules which provide initial schedules to the genetic algorithm. The genetic algorithm is being integrated with two simulation-based commercial scheduling packages. The genetic algorithm generates candidate schedules which are evaluated using the commercial packages. This evaluation provides a prediction of how well the shop will perform if this schedule were implemented. This generate-evaluate loop continues until the final solution is obtained.

We have also provided an preliminary EXPRESS data model for representing the status of the shop floor. This status provides the information the schedulers need to update their simulation models in order to do reactive scheduling. We have also provided exchange messages which could be used by commercial shop floor data collection system to provide that status.

We believe that this approach has the potential to solve real-world sequencing and scheduling problems quickly and to be usable on the factory floor. To benchmark the proposed approach, we have initiated a sequencing project with a US manufactuirng firm. We hope to have the results from that project in time for the conference.

ACKNOWLEDGEMENTS

The authors would like to thank the following people for their contributions to the development of the data model and the definition of the change messages. Stephanie Schank from AMP, Inc., Steven Duket and Douglas MacFarland from Pritsker Corporation, and Michael Rossi from AutoSimulation, Inc. In addtion, the authors which to recognize the Manufacturing Technology program of the US Navy, who have provided support funding for this project.

REFERENCES

BIEGEL, J. and DAVERN, J. Genetic Algorithms and Job Shop Scheduling, Computers and Industrial Engineering, 1190, vol. 19, no. 1, 81-91.

(8) DAVIS, L. Job Shop Scheduling with Genetic Algorithms, Proceedings on an International Conference on Genetic Algorithms and Their Applications, Carnegie-Mellon University, USA, 1985, 136-140.

GOLDBERG, D. Genetic Algorithms in Machine Learning, 1988, Addison-Wesley, Menlo Park, California, USA.

STARKWEATHER, T., WHITELY, D., and COOKSON, B. A Genetic Algorithm for Scheduling with Resource Consumption, Operations Research in Production Planning and Control, 1993, Springer Verlag, Berlin, 567-583.

YIH, Y. Trace-Driven Knowledge Acquisition (TDKA) for Rule-Based Real-Time Scheduling Systems, *Journal of Intelligent Manufacturing*, 1990, vol. 1, no. 4, 217-230.

De JONG, K. and SPEARS, W. Using Genetic Algorithms to solve NP-Complete Problems, Proceedings of the Third International Conference on Genetic Algorithms, Carnegie Mellon University, USA, 1989, 124-132.

JONES, A., RABELO, L., and YIH, Y. A Hybrid Approach for Real-Time Sequencing and Scheduling, International Journal of Computer Integrated Manufacturing, 1995, vol. 8, no.2, 145-154.

YIH, Y. and JONES, A. Candidate Rule Selection to Develop Intelligent Decision Aids for Flexible Manufacturing Systems, Operations Research in Production Planning and Control, 1993, Springer Verlag, Berlin, 201–217.

JONES, A. and RABELO, L. Integrating Neural Nets, Simulation, and Genetic Algorithms for Real–Time Scheduling, Operations Research in Production Planning and Control, 1993, Springer Verlag, Berlin, 550–566.

[16] GOLDBERG, D. and LINGLE, R., Alleles, loci, and the Traveling Salesman Problem, Proceedings of the of the International Conference on Genetic Algorithms and Their Applications, Carnegie Mellon University, USA, 1985, 162-164.

RUMELHART, D., HINTON, G., and WILLIAMS, R. Learning Internal Representations by Error Propagation, Parallel Distributed Processing, 1986, MIT Press, Cambridge, MA, 318-362.