

# Optimization of Enterprises Modeled by Rules

Huseyin Onur Mete<sup>1</sup>, Zelda B. Zabinsky<sup>2</sup> and Wolf Kohn<sup>3</sup>

<sup>123</sup>Industrial Engineering

University of Washington, Seattle, WA 98195-2650, USA

<sup>3</sup> Courant Institute of Mathematical Sciences

New York University 251 Mercer Street New York, N.Y. 10012-1185, USA

## Abstract

We develop a rule-based optimization approach. The rules are IF-THEN statements that allow domain experts to model their complex systems in a natural framework. We formalize the enterprise problem as minimizing multi-criteria objectives with respect to a set of business rules. We construct a finite state machine in which the rules correspond to state transitions. The states are evaluated by an output function under multiple objectives. The Pareto efficient frontier of states are called desired states. An optimization algorithm that identifies the Pareto optimal states is explored. We illustrate our methodology on a scheduling example.

## Keywords

Business rules, rule-based optimization, finite state automaton, Moore machine.

## 1. Introduction

We propose to develop a rule-based optimization approach to address the enterprise problem. Instead of traditional optimization tools, we incorporate rules that an expert can express by IF-THEN statements to understand and model the problem. The rule-based approach allows domain experts to model their complex systems in a natural framework. This research is to combine rule-based approaches with optimization for decision making under uncertainty.

In our approach, the optimization of an enterprise starts with modeling by rules, which provides an easy means of defining complex enterprise problems for domain experts. The main advantage of this approach is that it does not require advanced mathematical modeling abilities. Thus, the domain experts can reveal their system knowledge and experience by themselves easily during the modeling phase. We are going to incorporate the knowledge of rule-based production systems, which represent the knowledge in terms of multiple rules that specify the decision mechanism for different situations. The principal motive of employing rules is their similarity to the human cognitive process, so that they appear in a natural way that humans describe their problem. The structure and parameters of the problem can easily be elicited from the experts by simple IF (condition) THEN (action) representation of the rules. Moreover, there are usually several objectives that need to be achieved (minimal cost, timeliness, reliability of the solution) which can be incorporated by rules.

We need a method to involve rules in the optimization process. We propose using a finite state machine as a framework on which we design and run optimization algorithms. In this framework, the status of the system is represented by a state, which is defined by a vector of a set of attributes. The changes in the situation of the enterprise are revealed by transitions between states. Transitions are triggered by inputs. An applicable input moves the system from its current condition to a different state. The business rules correspond to the transitions in the finite state machine. In other words, when the IF part of a rule is satisfied, a transition will occur and move the finite state machine to the state defined in the THEN part of the rule. Thus, one of the major contributions of this research will be an automated mechanism that seamlessly constructs the appropriate finite state machine given the business rules.

We employ a Moore machine, which is an extension of a finite state machine by addition of an output function. Thereby, each state has an output vector calculated by the output function of the Moore machine. The output vector will provide a measure to evaluate and compare the states. We determine the desired states, the ones which we would like to end up with, according to this measure. This brings an optimization issue: We seek an optimal sequence of inputs that should be applied starting from the initial state in order to reach one of the desired states. Having the liberty of using a multidimensional output vector provides a mechanism to incorporate rules describing multiple objectives such as cost, timeliness, and reliability of the solution. We propose to establish the Pareto frontier as the set of desired states.

We demonstrate our rule-based optimization approach on a scheduling example. We want to schedule music groups in different cities at different performance times. The problem has multiple objectives (i.e. minimizing travel expenses,

minimizing empty performance times for cities). The requirements and objectives of scheduling music groups can be defined by rules. We present two ways of constructing Moore machines from rules. We seek a mechanism to find Pareto optimal states (i.e. non-beatable schedules of music groups) of the scheduling problem.

The traditional ways of modeling and optimization of enterprise problems like mixed-integer programming have several drawbacks in practice. They require mathematical modeling expertise and can be overly complicated when modeling complex systems. Moreover, the computation time to obtain a solution can be impractically long and have huge memory requirements. We propose a methodology to model the problem by business rules, which describe the problem in its natural domain and utilize a framework, finite state machine, on which we can design and run optimization algorithms. Thus, we state the optimization problem as starting from the current state of the system, finding the optimal sequence of inputs that lets the system reach desired states (i.e. Pareto optimal states).

## **2. Background and Literature Review**

We incorporate rules to define the optimization problem. Current rule-based methods have been successful in enterprise systems that need True/False evaluations, by checking and classifying phrases against rules, such as credit rating systems. These methods are often called expert systems. The rule based expert systems, which are the most popular type of expert systems, works as a production system in which rules encode expert knowledge. The knowledge is represented in terms of multiple rules that specify what should or should not be concluded in different situations instead of representing knowledge in a relatively declarative, static way (as a number of things that are true) [1].

The basic components of an expert system are the knowledge base, and the inference engine. The knowledge base contains the domain knowledge that is represented as a set of rules. The inference engine compares each rule stored in the knowledge base with facts contained in the database. When the if part of the rule matches a fact, the rule is fired and its then part is executed. There are two main methods of reasoning when using inference rules: forward chaining and backward chaining. Forward chaining, which is known as data-driven reasoning, starts from the known data and proceeds forward with that data by firing the topmost rule and adding a new fact in the database [2]. Backward chaining, which is the goal-driven reasoning, starts with a list of goals and works backwards to see if there is data which will allow it to conclude any of these goals [1].

The other major approach that combines rules and optimization is Constraint Programming (CP), which involves stating the relations between variables in the form of constraints. In the literature, CP is used for expressing constraints on the sequence of decision variables by regular constraints. In [4], global constraints are represented with a finite state machine and an associated layered graph structure for an algorithm to develop a MIP version of the constraints. Finite state machines (FSMs) provide a framework for the abstract design of algorithms which are implemented on a machine with a finite number of states representing the state of the algorithm [3]. They are concerned with taking an input, changing between internal states, and generating an output like all computing devices. FSMs must be described in terms of a basic set of features and operations which determine what the machine can do. Algorithms to solve problems can be developed by using this description. In a similar way, [5] solved employee time tabling problem by including global cardinality and regular constraints.

The simplest type of FSMs are finite automata, which consist of a finite set of states, an input alphabet, and a finite set of transitions for each state and symbol in the alphabet. A finite automaton reads each symbol in the string and causes a transition between states. If all symbols of the string have been read and the finite automaton is not in the finite state then the finite automaton is said to fail and the input string is not accepted. The finite automaton also fails if no transition exists for a symbol read.

Different than [4] and [5], we extend the abilities of finite automata by letting it output symbols. There are two extensions of finite automata in the literature: Moore machines and Mealy machines. Unlike finite automata, a Moore machine or a Mealy machine does not accept or reject input strings, instead they process them. In the Moore machines the outputs are determined by the current state alone and do not depend directly on the input, whereas Mealy machines provide the output based on its current state and an input. We incorporate Moore machines to be able to evaluate the outputs of the states. Thus, we can compare the states with multi-objectives and determine the desired ones. Thus, instead of solving a constraint satisfaction problem, we aim to design algorithms to reach the desired states of the automata from an initial state. The future work would include using non-deterministic automata to involve the uncertainty of the system by probabilistic transitions between states.

## **3. Constructing a Moore machine by business rules: Scheduling example**

We present a sample problem of scheduling music groups to illustrate the basics of constructing a Moore machine. First, we define the problem by business rules. Then, we develop a mechanism to construct a Moore machine of the

problem. We present two mechanisms, a complete Moore machine and a refined Moore machine.

Three music groups, each of which has a different musical style, are to be scheduled to perform in different cities, on different days and different performance times. The objective is to get a feasible solution that minimizes the travel and accommodation expenses of the music groups along with obtaining a desirable travel schedule for the musicians. The music groups that are going to be scheduled are Rock, Classical, and Jazz groups. We want to schedule them to perform in the concert halls in Seattle, Bellevue, Tacoma on Monday, Tuesday, and Wednesday. The groups can perform twice, matinee and soiree, per day.

The problem is defined by rules. Hard rules give the boundaries of the problem so that every solution of the problem must satisfy them, whereas soft rules represents the preferences for a solution. They can be broken to a degree. However breaking a soft rule incurs a cost or penalty. Usually, there is some limit to how much a soft rule may be broken, after that limit is reached, the soft rule behaves like a hard rule and cannot be broken. The rules of the music groups scheduling problem are as follows.

### **Hard rules**

1. At most one music group can be scheduled at a place at a time.
2. A music group cannot be scheduled in more than one place at a time.

### **Soft rules**

1. The music groups have different travel expenses. All music groups are assumed to be in Seattle initially (Sunday night) and need to return to Seattle at the end of the period (Thursday morning).
2. Accommodation and food expenses, which change by the city and the group should also be considered. The music groups will stay in a hotel in the city in which they performed last. Their accommodation expenses will be paid for Monday, Tuesday, and Wednesday nights. In addition, they will be paid for dinner after their matinee performances. The dinner costs 20% of the accommodation expenses at each city.
3. Each performance time at each city should be filled. Empty blocks will cause a penalty. We never want a completely empty schedule, thus, this rule can be converted to a hard rule to disallow all empty blocks (or a fixed number).
4. Each music group should hold at least one performance in every place. Skipping cities has a penalty.
5. A music group should not be scheduled in two different cities in a day. Thus, there will be a penalty for changing cities in a day.

### ***Selected instances of the problem***

**Problem I:** In this simplified instance we only have two cities (Seattle and Bellevue), one day (Monday), two performance times (matinee and soiree) and two music groups (a rock group and a classical group) to be scheduled.

**Problem II:** Extend the first problem by one day; two cities (Seattle and Bellevue), two days (Monday and Tuesday), two performance times at each day (matinee and soiree) and two music groups (a rock group and a classical group).

**Problem III:** This is the complete instance of the problem that includes three cities (Seattle, Bellevue and Tacoma), three days (Monday, Tuesday and Wednesday), two performance times at each (matinee and soiree).

### **3.1. Developing a mechanism to construct a Moore machine from rules**

We develop an automated mechanism to construct the Moore machine of the problem. A Moore machine is a 5-tuple  $M = \{Q, \Sigma, \Delta, \beta, Y\}$ , where  $Q$  is the finite set of states,  $\Sigma$  is the input alphabet and  $\Delta$  is the set of transitions. Inputs behave like functions (i.e.  $\sigma \in \Sigma, \sigma : Q \rightarrow Q$ ) in taking one state to another one. A transition  $\delta \in \Delta$  is defined by an ordered triple  $\delta = (q_1, q_2, \sigma)$ , that is,  $q_1 \in Q, q_2 \in Q$ , and  $\sigma \in \Sigma$ , a transition to state  $q_2$  is created by applying the input  $\sigma$  to state  $q_1$ . In Moore machines, the outputs are determined by the current state alone and do not depend directly on the input as opposed to Mealy machines [3].

In connection to a rule-based approach, each rule corresponds to a transition. In this case, the IF part of the rule is satisfied by having a state  $q_1$ , and the execution of the rule brings the Moore machine to state  $q_2$  which is defined in the THEN part of the rule. The output function,  $\beta : Q \rightarrow Y$  is used to evaluate the output of the state, which provides a measure to evaluate the states. Thereby, we can determine the desired states that we would like to reach when we

start from an initial state. Determining the optimal sequence of inputs that will start from an initial state and reach a desired state is the issue of constructing an optimization algorithm to be run on a Moore Machine.

We developed two ways of constructing Moore machines for the sample problem. In the first one, we derived the complete list of states by making all possible assignments of music groups to performance times. Then, out of the complete list of states we determined the desired states by Pareto efficiency. The number of the states explodes even for small problems. Therefore, we developed another way of constructing the Moore machine based on eliminating schedules that would not yield non-Pareto frontier results. This method significantly reduces the number of states in the Moore machine.

The elements of the Moore machine,  $M = \{Q, \Sigma, \Delta, \beta, Y\}$  are explained below.

**The set of states** ( $q \in Q$ ): States are defined as the schedule of music groups to performance times and locations. The starting state of the Moore Machine is the empty schedule, which has no assignments in any performance time.

The number of states that satisfy the first hard rule is  $\sum_{i=0}^T \binom{T}{i} G^i$  where  $T$  represents the total number of performance times (6 in Problem III) and  $G$  represents the number of music groups (3 in Problem III). For Problem I, we have 81 states that satisfy the first hard rule whereas Problem II has 6,561 states, and Problem III has 68,719,476,736 states. The second hard rule reduces the number of states. Problem I has 49 states that satisfies both hard rules and Problem II has 2,402 states. Although, the second hard rule reduces the number of states in Problem III, it is still an impractical number to calculate.

**Input set** ( $\sigma \in \Sigma$ ): Each input represents making an assignment to a specific performance time. For example, assigning the jazz group to Bellevue for the Monday Matinee is an input for Problem III. For Problem I, there are eight possible assignments, or inputs, that can be applied.

The cardinality of the input sets of the problems is given by  $|\Sigma| = G * T$ . Thus, the number of inputs for Problems I, II, and III are 8, 16, and 54 respectively.

**Transitions** ( $\delta \in \Delta$ ): In the Moore machine, we have transitions for each state and input. They are ordered triples  $(q_1, q_2, \sigma)$  where  $q_1, q_2 \in Q$  and  $\sigma \in \Sigma$  and  $q_2$  is uniquely determined by  $q_1$  and  $\sigma$ . For the given scheduling problem, the inputs create schedules by starting from a blank schedule and making assignments to performance times one by one.

**Output function** ( $\beta : Q \rightarrow Y$ ): The output function maps each state to an output vector. For the scheduling problem the output of a state is the vector of costs, or penalties, due to the violation of soft rules. Thus,  $\beta = [\beta_1, \beta_2, \beta_3, \beta_4, \beta_5]^T$ , where

$\beta_1 : Q \rightarrow \mathbb{R}$ : the function that calculates the travel expenses

$\beta_2 : Q \rightarrow \mathbb{R}$ : the function that calculates the accommodation and food expenses

$\beta_3 : Q \rightarrow \{1, 2, \dots, T\}$ : the function that gives number of empty blocks

$\beta_4 : Q \rightarrow \{1, 2, \dots, G * L\}$ : the function that gives number of skipped cities for each group

$\beta_5 : Q \rightarrow \{1, 2, \dots, D * L\}$ : the function that gives number of changing cities in a day

and  $D$  and  $L$  represent the number of days and locations respectively.

**Output set** ( $y \in Y$ ): The output set is the target set of the output function. The output set of the scheduling problem is a 5-dimensional vector space. Thus,  $y = [y_1, y_2, y_3, y_4, y_5]^T \in Y$ , where  $y_i$  represents travel expenses, accommodation and food expenses, number of empty blocks, number of skipped cities for each music group, number of changing cities in a day, for  $i = 1, 2, 3, 4, 5$ , respectively.

### **Desired states - Pareto efficiency**

Among the states of the Moore machine, we would like to determine the ones that give preferable output values. The output function with multi-dimensions leads us to a multi-criteria comparison of states. For instance, the empty schedule has zero travel expenses, whereas it has  $T$  empty performance times.

In order to determine the desired states by eliminating the undesirable ones, we utilize the idea of Pareto efficiency. Given a state, a movement to another state that can make at least one dimension of the output vector better off, without making any other dimension worse off, is called a Pareto improvement. A state is Pareto efficient or Pareto optimal when no further Pareto improvements can be made. The Pareto frontier is the set of states that are all Pareto efficient. To clarify, let us take two states, say  $q'$ , and  $q''$ . State  $q'$  dominates state  $q''$  when  $q' \neq q''$  and  $\beta_i(q') \leq \beta_i(q'')$ . The Pareto frontier,  $P(Y)$ , is the set of states that cannot be dominated by any state in  $Q$ . That is,

$$P(Y) = \{q' : \{q \in Q : \beta_i(q) \leq \beta_i(q') \ i = 1, 2, \dots, 5, q' \neq q\} = \emptyset\}. \quad (1)$$

In this respect, 10 of the 49 states of Problem I are in the Pareto frontier where Problem II has 50 Pareto optimal

states out of 2,402 feasible states. Since the number of states explodes, it is impractical to determine the number of feasible states for Problem III. Thus, we seek a more efficient way of constructing the Moore machine in the next section.

### 3.2. Refined Moore machine

Due to the explosion in the number of states in the Moore machine given above, determining the set of desired states (i.e. Pareto frontier) requires an unacceptable amount of time and memory. Therefore, we developed a refined method for constructing the Moore machine based on eliminating states that would give non-Pareto frontiers. The elements of the new Moore machine are given below.

**The set of states** ( $q \in Q$ ): States are defined as the sequential assignments to performance times. That is, the state definition includes the number of performance times considered, the current location of music groups and the visited cities information for each music group in addition to the schedule of music groups to performance times and locations.

**Input set** ( $\sigma \in \Sigma$ ): Each input takes a state with performance times considered up through  $k$  and makes an assignment for performance time  $k + 1$ , thereby producing a new state. For example, given a state with Monday matinee considered, assigning a jazz group to Seattle and a classical group to Bellevue for the next performance time (Monday soiree) is an input for Problem I.

**Transitions** ( $\delta \in \Delta$ ): The transitions  $\delta = (q_1, q_2, \delta)$  are only defined for states  $q_1$  and  $q_2$  that differ in their performance time by one. That is, if  $k$  performance times have been considered for state  $q_1$ , only states with performance time  $k + 1$  are valid in the transition triple.

**Output function** ( $\beta : Q \rightarrow Y$ ) **and Output set** ( $y \in Y$ ): The output function and the output set remain the same as given in Section 3.1.

#### *Pareto frontier of the refined Moore machine*

We include an elimination process for applying inputs to the states so the inputs will not be applied to the states that would lead to non-Pareto frontier states.

**Theorem 1** *Let  $q_1$  and  $q_2$  be two different states with performance times  $k_1$  and  $k_2$  considered, respectively. Suppose the following three conditions hold,*

- i)  $k_1$  equals  $k_2$ ,
- ii) *the current location at time  $k_1 = k_2$  of each music group is the same for both  $q_1$  and  $q_2$  (the current location of a music group is its last assigned location),*
- iii) *each music group has visited the same cities in both  $q_1$  and  $q_2$ .*

*If  $q_1$  dominates  $q_2$ , that is,*

$$\beta_i(q_1) \leq \beta_i(q_2) \quad \forall i \in \{1, 2, \dots, 5\} \text{ and} \quad (2)$$

$$\exists i \in \{1, 2, \dots, 5\} \text{ such that } \beta_i(q_1) \neq \beta_i(q_2) \quad (3)$$

*then, any further applications of inputs to  $q_2$  cannot give a Pareto frontier state.*

The proof of the theorem above is based on the fact that applying inputs to any two states at the same performance time and with the same current location and visiting information, changes the elements of the output function at the same rate, therefore, applying any input to a dominated state will always result in again a dominated state. Theorem 1 implies that we can eliminate states after each  $k$  and still obtain all Pareto optimal final states.

For the comparison of the two approaches, it is important to note that the former way can solve Problem II, which is relatively a small problem, in 20 minutes, whereas it takes less than a second by the refined approach. The refined version can solve a problem of two groups, two cities and 20 performance times (10 days) in 60 minutes, which is practically impossible to solve using the complete approach. However, the refined approach can still not solve Problem III in a reasonable time, and we are working on developing more refinements to the Moore machine approach.

## 4. Conclusion

We are exploring a methodology to solve a multi-criteria enterprise optimization problem defined by business rules. This approach provides an opportunity of expressing the problem by business rules in a natural framework. We seek a

method to construct a finite state automaton, Moore machine, of the problem; thereby having a framework to analyze and solve the business problem. The efficiency of the solution depends on the quality of the automata. Therefore, we discuss the methods to reduce the size of the automata and provide two ways of this construction: complete Moore machine and refined Moore machine on the music scheduling example. We would like to develop automatic and more efficient ways of creating automata from business rules and optimization algorithms to be run on them.

## **References**

- [1] Giarratano, J.C. and Riley, G.D., 2005, Expert systems principles and programming, 4th edition, Thomson Course Technology, Boston, MA.
- [2] Negnevitsky, M., 2002, Artificial intelligence: a guide to intelligent systems, Addison-Wesley, New York.
- [3] Hardy, Y. and Steeb, W.H., 2001, Classical and quantum computing with C++ and Java simulation, Boston, MA.
- [4] Cote, M.C., Gendron B. and Rousseau L.M., 2007 “Modeling the regular constraint with integer programming,” Proceedings of the International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Brussels, Belgium.
- [5] Demasse S., Pesant G. and Rousseau L.M., 2005 “Constraint programming based column generation for employee timetabling,” Proceedings of the International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Prague, Czech Republic.