*Artificial Intelligence and Language Processing*

*Peter Friedland Editor*

# Integer Programming vs. Expert Systems: An Experimental Comparison

*Expert system and integer programming formulations of an NP-complete constraint satisfaction problem are contrasted in terms of performance, ability to encode complex preferences, control of reasoning, and supporting incremental modification of solutions in response to changing input data.*

**Vasant Dhar and Nicky Ranganathan**

Virtually all decision making situations involve constraints. What distinguishes various types of problems is the form of these constraints. Depending on how the problem is visualized, they can arise as rules, data dependencies, algebraic expressions, or other forms.

Constraint satisfaction problems (CSPs) have been studied extensively in the operations research (OR) and artificial intelligence (AI) literature. In OR formulations constraints are quantitative, and the solver (such as the Simplex algorithm) optimizes (maximizes or minimizes) the value of a specified objective function subject to the constraints. In contrast, AI research has focused on inference-based approaches with mostly symbolic constraints. The inference mechanisms employed include theorem provers, production rule interpreters, and various labeling procedures such as those used in truth maintenance systems.

It has been apparent for some time that there is a close relationship between logical and quantitative inference. Dantzig [2] showed how logical propositions could be expressed quantitatively using Boolean variables together with algebraic operators. In this way, symbolic constraints can be modeled in terms of an integer programming formulation. For example, the constraint $\neg x_1, x_2 \rightarrow x_3$ is equivalent to the clause "$x_1$ or not-$x_2$ or $x_3$" where each $x_i$ is a propositional variable. A clause can be expressed as an inequality; the above clause is equivalent to

$$x_1 + (1 - x_2) + x_3 \geq 1$$

where the truth values true and false are denoted by 1 and 0, respectively. In general, as has been noted independently by Hinton [7] and Hooker [8], a clause can be expressed as the following inequality:

$$c_1x_1 + \cdots + c_nx_n \geq 1 - n(c)$$



where the elements $c_1, c_2, \ldots, c_n$ are elements of a row vector $c$, $x_1, x_1, x_2, \ldots, x_n$ are elements of a column vector $x$, and $n(c)$ is the number of negative elements in $c$. Each $c_i$ is 1, 0, or $-1$, indicating whether $x_i$ appears, does not appear, or $\neg x_i$ appears in the clause respectively. The above notation is from Hooker [8].

In concrete terms, the symbolic/quantitative equivalence means that many constraint satisfaction problems can in principle be modeled using symbolic inference techniques employed in expert systems or the quantitative techniques of OR. In practice, there are pros and cons to both approaches. One of our objectives has been to study the tradeoffs involved in detail.

Over the last year, several researchers of the Reasoning Architectures group at the Microelectronics and Computer Technology Corporation (MCC) Artificial Intelligence Laboratory have collaborated with an experienced department head at the University of Texas in analyzing the problem of planning the assignment of faculty to courses, and revising plans as assumptions (about faculty availability and enrollments) change. The problem is of interest for several reasons. It characterizes many types of planning problems where solutions are assumption based or defeasible and must be revised in light of a changing reality. The problem also involves a diverse variety of constraints, making it a good canonical problem for analyzing different modeling approaches.

Based on an extensive knowledge engineering effort over the last year, an expert system has been built and implemented in PROTEUS [14]. The expert system (called RACS—Revisable Academic Course Scheduler) consists of a heuristic rule-based problem solver, which contains the expert's knowledge, and a truth maintenance system (TMS), which maintains a contradiction-free database of assignments.

In this article, we present an integer programming formulation corresponding to the expert system and report on our experiences with running this model. We are interested in studying whether the integer programming model can provide the results that derive from the reasoning process of the expert, whether it can be modified easily to accommodate changes, and how general purpose integer programming techniques do in terms of computational performance.

The remainder of this article is organized as follows. In the next section we provide a mathematical formulation of the problem and describe its general structure. This is followed by a description of the problem-solver/TMS architecture that constitutes the expert system. We then describe the architectural components of the IP implementation and analyze the results from running this model, comparing them with an expert's decisions when presented with the same problem data. In doing so, we classify some of the limitations of the mathematical model and describe why the problem-solver/TMS combination does not suffer from these problems (although it has other drawbacks). We conclude with a list of short- and long-term issues that must be addressed in order to design more expressive and flexible reasoning systems.

## THE INTEGER PROGRAMMING FORMULATION

Generating a course schedule requires assigning faculty to courses while taking into consideration a variety of constraints and preferences. Constraints pertain to bounds on the number of sections of courses, minimal faculty teaching requirements, dependencies (i.e., whoever teaches "*Introduction to Logic*" must also teach "*Advanced Logic*"), etc. Preferences are expressed by teachers for desired courses and alternatives. In addition, the scheduler (chairman) expresses preferences in deciding how to deal with conflicts that arise in synthesizing the plan. After courses have been assigned, class rooms and times are scheduled. We do not deal with the latter problem in this article.

The structure of the problem data is presented in Figure 1. There are three classes of data involved: teacher, course, and term. There are three types of teachers (three subclasses), each with specific instances. Likewise, there are two types of courses and instances of these two types, and two terms—fall and spring. Constraints are defined at all levels of abstraction, in terms of the various classes and instances.

The formulation of the problem we present is based on a detailed problem description that appears in Petrie et al. [12]. Many of the constraints in the IP formulation have been derived from the rules in the expert system. The IP formulation is best presented by first defining all subscripts, variables, and coefficients, followed by the constraints and objective function.

### Subscripts

$i = 1, 2, \ldots, n$ instructors
$j = 1, 2, \ldots, m$ courses
$t = 1, 2$ terms (fall or spring)
$k = 1, 2 \ldots, p$ categories of courses (for balancing curriculum)

### Variables

$$x_{ijt} = \begin{cases} 1 & \text{if faculty } i \text{ is assigned to course } j \text{ in term } t \\ 0 & \text{otherwise} \end{cases}$$

### Coefficients

$\rho_j$ = load factor of course $j$ (depending on its size and type)
$T_i$ = teaching load requirement (TL) for teacher $i$
$lb_{jt}$ = lower bound on sections of course $j$ to be taught in term $t$
$ub_{jt}$ = upper bound on sections of course $j$ to be taught in term $t$
$lb_j$ = lower bound on total sections of course $j$ to be taught in year
$ub_j$ = upper bound on total sections of course $j$ to be taught in year
$y_j$ = number of sections of course $j$ to be taught in the entire year
$G_t$ = the number of graduate level courses normally offered during term $t$
$c_{ijt}$ = cost of assigning teacher $i$ to course $j$ in term $t$.

This can take on three values:

$$c_{ijt} = \begin{cases} 0 & \text{if course } j \text{ is desired by faculty } i \text{ in term } t \\ M\text{, a small number} & \text{if } j \text{ is an explicit alternative} \\ Z\text{, a large number} & \text{if } j \text{ is an implicit alternative or} \\ & \text{if faculty } i \text{ can teach course } j \text{ in term } t_1 \neq t \end{cases}$$

Actually, another number is added to the value of $c$ to incorporate the flexibility of a teacher in terms of his capability and propensity to teach alternative courses. Specifically, teachers are ranked on a scale of 1 to 5 (both inclusive) where a 1 expresses least flexibility. The combined number is the cost coefficient and expresses a "penalty" associated with assigning faculty $i$ to course $j$ in term $t$. We shall discuss this further shortly.

$$g_j = \begin{cases} 1 & \text{if } j \text{ is a graduate level course} \\ 0 & \text{otherwise} \end{cases}$$

$$u_j = \begin{cases} 1 & \text{if } j \text{ is an upper division course} \\ 0 & \text{otherwise} \end{cases}$$

$$l_j = \begin{cases} 1 & \text{if } j \text{ is a lower division course} \\ 0 & \text{otherwise} \end{cases}$$

$$w_j = \begin{cases} 1 & \text{if } j \text{ is a writing course} \\ 0 & \text{otherwise} \end{cases}$$

$$f_i = \begin{cases} 1 & \text{if } i \text{ is a faculty member} \\ 0 & \text{otherwise} \end{cases}$$

$$C_{jk} = \begin{cases} 1 & \text{if course } j \text{ is in category } k \\ 0 & \text{otherwise} \end{cases}$$

$$t_{ijt} = \begin{cases} 1 & \text{if course } j \text{ is proposed as a tutorial by } i \text{ in term } t \\ 0 & \text{otherwise} \end{cases}$$

$$AB_j = \begin{cases} 1 & \text{if course } j \text{ has the } A\text{–}B \text{ sequence} \\ 0 & \text{otherwise} \end{cases}$$

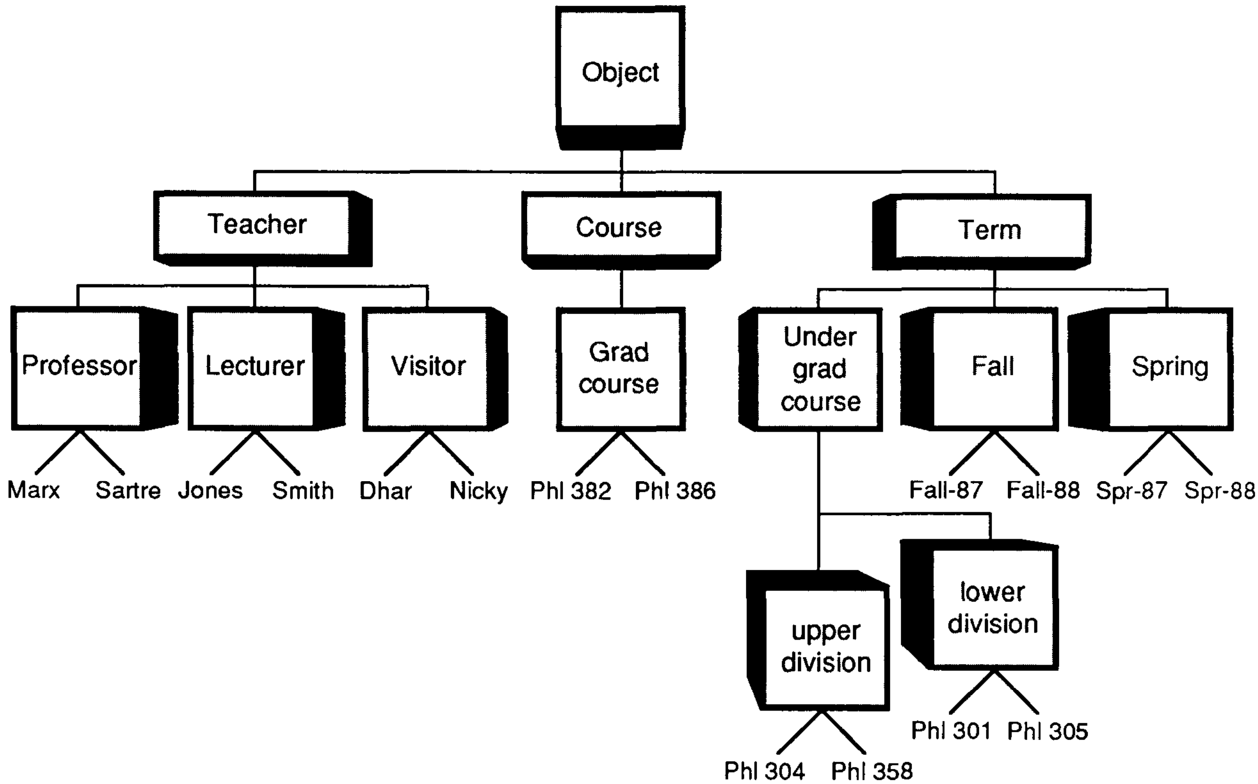


**FIGURE 1. Data Hierarchy**

**Constraints**

1. Number of teachers assigned to a course in each term should be between the lower and upper bounds on the number of sections of that course:

$$lb_{jt} \leq \sum_{i=1}^{n} x_{ijt} \leq ub_{jt}$$

$$\text{for } 1 \leq j \leq m, \quad \text{and } t = 1, 2$$

2. The total number of sections of course $j$ taught in the year should be:

$$lb_j \leq \sum_{t=1}^{2} \sum_{i=1}^{n} x_{ijt} \leq ub_j$$

$$\text{for } 1 \leq j \leq m, \quad \text{and } t = 1, 2$$

3. Each teacher must satisfy some minimal teaching load:

$$\sum_{j=1}^{m} \rho_j x_{ijt} \geq T_i$$

$$\text{for } i = 1, 2, \ldots, n, \quad \text{and } t = 1, 2$$

4. Only professors can teach graduate courses:

$$x_{ijt} \leq 1 - (1 - f_i)g_j$$

$$\text{for } i = 1, 2, \ldots, n, \quad \text{for } j = 1, 2, \ldots, m \quad \text{and } t = 1, 2$$

5. No professor can teach more than $G$ graduate courses per year:

$$\sum_{j=1}^{m} \sum_{t=1}^{2} x_{ijt} g_j \leq G \quad \text{for } i = 1, 2, \ldots, n$$

6. Course sequence continuity ($A$–$B$) constraint:

$$AB_j(x_{ij2} - x_{ij1}) \leq 0$$

$$\text{for } i = 1, 2, \ldots, n, \quad \text{and } j = 1, 2, \ldots, m$$

Constraint 6 says that someone can only teach a $B$ course in spring if they taught the $A$ course in the fall. If someone *must* teach the $B$ course if they taught the $A$ course in fall, the inequality must be changed to an equality.

7. There must be at least $U$ upper division writing courses offered each term:

$$\sum_{j=1}^{m} x_{ijt} w_j u_j \geq U$$

$$\text{for } i = 1, 2, \ldots, n \quad \text{and } t = 1, 2$$

Similar constraints are formulated for lower division writing courses and the number of graduate courses that must be offered each term.

### Objective Function

A variety of objective functions can be formulated for this problem depending on the primary goal. One reasonable goal is to maximize fit, or minimize deviations from teachers' desired courses. To express this, the objective function is:

$$\text{Minimize } z = \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{t=1}^{2} c_{ijt} x_{ijt}$$

This objective function states that the good of the many outweights the good of the few.

### Problem Structure

If one considers courses as "producing" sections for which there are maximum and minimum supplies (capacities) and teachers as "demand" points, the problem can be viewed as a transportation problem. In fact, with constraints 1 and 2 are the objective function, the problem is a classical transportation problem. This problem has a nice property in that it produces an integer solution for integer inputs. Thus the problem of fractional assignments does not arise.

Unfortunately, the third constraint, relating course load factors to teaching load requirements, is a knapsack problem constraint (see page 65 of Garey and Johnson [6] for a description of this problem). This destroys the integer solution property. More importantly, the problem becomes NP-complete and its solution time varies extremely unpredictably.

Actually, the real problem is more complex in one other way. On occasion, a teacher may teach more than one section of a course. Also, a few courses have sections of different sizes (i.e., large, double-large) and therefore have different load factors associated with them. Modeling this requires modifying the decision variable to include another subscript for course sections. If this is done, the number of decision variables increases significantly thereby increasing the overall complexity of an already difficult to solve problem. Specifically, the problem becomes a bin packing problem which is known to be NP-hard [6]. The expert deals with this complexity by assuming default section sizes in making assignments to come up with a rough initial (usually infeasible) solution, and then for each teacher either increasing assigned section sizes in order to satisfy a teaching load requirement, or assigning additional sections of a course to a teacher if they have been explicitly requested.

The IP formulation deals with this problem in a rather inelegent, though practical way. Specifically, since relatively few courses (e.g., PHL 304) have sections of different sizes (PHL304Large, PHL304Double-Large), these different sized sections are treated as different courses with appropriate load factors. Similarly if a teacher requests more than one section of a course, they are treated as different courses (e.g., PHL304-1, PHL304-2). In effect, such courses (like PHL304 in this example) are treated as object classes; constraints on the number of sections of such a course are then stated in terms of instances (PHL304Large, PHL304Double-Large etc.) of the class.

## EXPERT SYSTEM ARCHITECTURE

In contrast to the IP model, the expert system has no global objective function that guides it toward an optimal solution. Rather, the process of finding a solution involves a series of local decisions that are most preferred at each point in the problem solving process.

The expert system consists of two components, a problem-solver that employs a generate-and-test strategy, and a TMS. The problem solver is essentially a production system consisting of rules whose patterns are matched by assertions (propositions) in a global database. Each assertion is a problem-solver datum such as an assignment of a value to a variable (i.e., a decision) or an operation that can result in an assignment. The problem data is represented in terms of an ISA object hierarchy corresponding to the structure in Figure 1. Problem constraints are stated in terms of predicates that reference objects in the hierarchy. For example, if the default teaching load for a professor is 9 credits, this would be represented by the proposition "(TL ?F:Professor 9)" where ?F denotes a variable that will unify with object instances of type Professor. In this case, TL is a binary predicate, which is a slot in the Professor object type. With the above proposition, unless the TL for an instance of Professor is otherwise specified, its value will be 9. Similarly, the proposition "(max-sections PHL304 ?T:Term 10)" states that the maximum number of sections of PHL304 in any term is 10. For details about the frame-based representation of PROTEUS, the reader is referred to [15].

Apart from the data that are specified as part of the problem description, the problem-solver creates data corresponding to the assignments in generating the solution. Each problem-solver datum has a justification associated with it. The justification encodes the reasons for belief in the datum. The task of the TMS is to ensure that the global database is contradiction-free, that is, the data are logically consistent (their truth values do not result in logical contradictions).

Before describing the problem-solver it is expedient to explain how the TMS works. For an overview of truth maintenance systems, the reader should refer to survey articles by Reinfrank [16], McAllester [11], or Dhar [4]. However, the fundamental concepts and mechanics underlying the TMS should be clear in the following description.

### The Truth Maintenance System

There are two types of TMSs, *justification-based* [5] and *assumption-based* [3]. We shall concern ourselves with the former type. Further, the discussion is in terms of a Doyle-style TMS since this is the type of TMS used in the expert system.

In a TMS, every datum has a *support status* associated with it; a status of IN indicates that the datum is currently believed whereas an OUT indicates disbelief.

The values IN and OUT are computed via *justifications* associated with the datum. Each justification has two parts, an *inlist* and an *outlist*. A justification is considered *valid* if it evaluates to *true*, that is, if each datum in its inlist is IN and each datum in its outlist is OUT. Ultimately, all data depend on "ground" level justifications of two types, *premises* and *assumptions*. In a premise justification, the inlist and outlist are both empty. This type of justification is always valid. An assumption justification has a non-empty outlist, that is, its belief is justified by a lack of belief in some other datum. Such a justification is considered *non-monotonic* since its validity depends on a lack of belief in some other datum. Finally, deductive justifications are those in which the outlist is empty. A datum can have more than one justification associated with it. A datum is IN if it has at least one valid justification, otherwise it is OUT. A datum without a justification (an empty justification) is OUT.

To illustrate, consider the following example. Upper case symbols denote functions and lists of lower case symbols refer to data:

```
Datum: (assign Sartre PHL304)
Justification:
  (AND (INLIST (wants Sartre PHL304)
               (available PHL304))
       (OUTLIST ()))

Datum: (assign Kant PHL304)
Justification:
  (AND (INLIST (wants Kant PHL304)
               (available PHL304))
       (OUTLIST ()))

Datum: (available PHL304)
Justification:
  (AND (INLIST ())
       (OUTLIST
       (max-sections-satisfied PHL304)))

Datum: (wants Sartre PHL304)
Justification:
  (AND (INLIST ())
       (OUTLIST (on-leave Sartre)))

Datum: (wants Kant PHL304)
Justification:
  (AND (INLIST ())
       (OUTLIST (on-leave Kant)))

Datum: (max-sections-satisfied PHL304)
Justification: ()

Datum: (on-leave Sartre)
Justification: ()

Datum: (on-leave Kant)
Justification: ()
```

The example can be visualized in terms of the graph in Figure 2 which shows the justifications of data referenced in the previous justifications. Each circle corresponds to a justification, with an arrow pointing to the justified datum, positive arcs connected to the elements of the inlist and negative arcs connected to the elements of the outlist. The datum "(assign Sartre PHL304)" has a two element inlist. The datum "(wants Sartre PHL304)" has an empty inlist and a non-empty outlist, which is a non-monotonic justification. The datum "(on-leave Sartre)" has an empty list of justifications and is therefore OUT. If this datum were to acquire a valid justification (if Sartre were to go on leave), its support status and that of those that depend on it must be reevaluated. Specifically, the "(wants Sartre PHL304)" would become OUT, also causing "(assign Sartre PHL304)" to go OUT.

Actually, in reevaluating the belief status of the data, referred to as *reason maintenance*, what the TMS really does is to ensure that the data as a whole satisfies two properties, *consistency* and *well-foundedness*. In a consistent state each datum with at least one valid justification is IN and each one without a valid justification is OUT. A state is well-founded if no set of beliefs is mutually monotonically dependent; in terms of a labeled network such as that in Figure 2, this means that there is no set of arcs from a node to itself all of which are labeled positively.

In the expert system, the TMS works in conjunction with the problem solver as follows. Each problem-solver action is communicated to the TMS at which point the TMS executes a constraint satisfaction procedure to ensure consistency and well-foundedness. Essentially, this involves updating the justifications associated with each datum such that the two conditions are satisfied. A problem-solver action can lead to a constraint violation which is recorded in the network as a contradiction, that is, a special node called *contradiction* becomes IN. When this is detected by the TMS, it tries to compute a new labeling that makes the contradiction OUT. Control then passes back to the problem solver, and the cycle repeats. We shall later describe in more detail the workings of the TMS in the context of an example.

**The Problem Solver**

The problem solver is a production system consisting of a set of rules, some of which are designed to implement a generate-and-test strategy. It should be noted that there are alternative AI approaches to solving constraint satisfaction problems that employ a somewhat different problem solving strategy, namely, the principle of least commitment (Marr, [9]; Stefik, [17]; Waltz [18]). In least commitment, the solver avoids making a choice that might have to be undone later. In effect, the objective is to avoid backtracking that is characteristic of the generate-and-test approach. The least commitment approach usually involves a problem solver iterating over "states" or variables that have several possible associated interpretations that become successively constrained with each cycle. This approach has been applied most effectively in computational models of vision ([9, 18]). A feature of such problems is that there is sufficient new information about each state in each

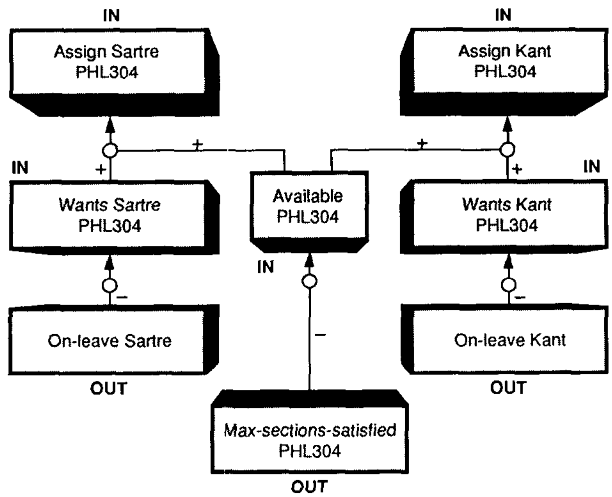


**FIGURE 2. A Consistent Well-Founded State**

iteration so that generate-and-test is in fact a poor strategy. The assignment problem we have modeled can be viewed in this framework as one where the possible interpretations for each state (teacher) are already specified at the outset as a constrained ordered set of courses (the course repertoire). Therefore, in order to generate a solution, the problem-solver uses a generate-and-test strategy based on this ordering.

In the following paragraphs we describe the expert system architecture using a simplified set of rules that play a role in one part of the problem solving process. The rules employ a LISP-like prefix notation that is used in PROTEUS [13]. In the following rules a question mark followed by a symbol denotes a variable, i.e. *?xxx* is the variable *xxx*. Each list in a rule consists of a pattern (a form) that is matched against a datum in the global database. There are two types of rules, forward and backward. When the problem solver has a proposition that it is trying to prove to be true, it attempts to do so via a backward rule. A "←" symbol denotes a backward rule; the form appearing before it is the consequent and the ones after it are antecedents. In trying to prove a goal, the assertions that match the antecedents become subgoals. Proving all of these subgoals completes the proof of the original goal. This process is known as backward chaining.

The symbol "→" denotes a forward rule. The forms appearing before it are antecedents, the others are consequents. The forward rule "fires" when assertions match its antecedent. When this happens, the datum (instance) corresponding to its consequent is added automatically to the global database. This process is known as forward chaining.

In the following rules, the variables used in the patterns are as follows:

*?prof* refers to the current professor under consideration.
*?sem* refers to a semester (either fall or spring).
*?course* refers to the course that is under consideration for assignment.
*?p-cycle* refers to the list of professors who have already been considered for *?course.*
*?c-cycle* refers to the list of courses that have already been considered for *?prof.*

The following backward rule states that the goal of making a course *?course* a chosen course for professor *?prof* in semester *?sem* can be achieved if that course is actually desired by the professor unless that course has already been considered (and presumably failed) for that professor:

```
(Mark-desired-course-as-chosen
              ;;name of this rule, rule 1
  (chosen-course ?prof ?course
                 ?sem ?c-cycle)
  ←
  (desired-course ?prof ?course
                  ?sem ?c-cycle)
  (unless (element ?course ?c-cycle)))
```

The following forward rule would fire whenever there is an assertion stating that courses be assigned to a professor:

```
(Select-Course    ;;rule 2
  (assign-courses-to-prof
    ?prof ?sem ?p-cycle ?c-cycle)
  (chosen-course ?prof ?course
                 ?sem ?c-cycle)
  →
  (attempt-to-satisfy
    ?prof ?course ?sem ?p-cycle
    ?c-cycle))
```

In the previous rule, the datum corresponding to the consequent would have the two data items matching the first two antecedent forms on its inlist.

The remaining rules can be interpreted similarly. Each form is accompanied by a comment whenever necessary.

```
(Semester-Succeeds    ;;rule 3
  (attempt-to-satisfy
    ?prof ?course ?sem ?p-cycle
    ?c-cycle)
  (unless
    (unacceptable-for-semester
      ?sem ?prof ?course ?p-cycle
      ?c-cycle))
  →
  (successful
    ?prof ?course ?sem ?p-cycle
    ?c-cycle))
                    ;;mark as successful
```

In this rule, a datum corresponding to the consequent would have the datum matching the first form on its inlist and the datum matching the form in the "unless" part on its outlist. In effect, it would have a non-monotonic justification.

```
(Semester-Fails     ;;rule 4
  (attempt-to-satisfy
    ?sem ?prof ?course ?p-cycle
    ?c-cycle)
     ;;If this semester is not acceptable
  (unacceptable-for-semester
    ?sem ?prof ?course ?p-cycle
    ?c-cycle)
  →
(failed-semester
  ?sem ?prof ?course ?p-cycle
  ?c-cycle))

(Contradiction-detection-type-1
      ;;rule 5
  (failed-semester
    ?sem ?prof ?course ?p-cycle
    ?c-cycle)
  →
(CONTRADICTION))
```

Constraint knowledge is also encoded as rules. These are used to enforce teaching loads, bounds on the number of sections, and other requirements such as those stated in the IP formulation. For example, the following rule (in conjunction with other rules such as rule-3 and rule-4) enforces the upper bound on the number of sections for a course during a particular semester:

```
(Course-section-check    ;;rule 6
  (unacceptable-for-semester
    ?sem ?prof ?course ?p-cycle
    ?c-cycle)
  ←
  (semester-max-constraint-satisfied
    ?course ?sem))
```

In addition to the assignment and constraint checking types of rules, the system also contains rules for contradiction resolution. When a contradiction arises, the TMS first finds a *culprit* to be made OUT to resolve the contradiction. The process of finding a culprit is a recursive one that involves determining the ground level data (the assumptions) that support the contradiction. The contradiction is resolved by invalidating all valid justifications of one of its supporters, called the *culprit*. This involves justifying some belief on the outlist of the justification being invalidated. For example, if an attempted assignment fails, one way to resolve the problem is to exchange that (chosen) course with an alternative course that that professor can teach. The following rule shows the use of a predicate, FIX, that is designed to do contradiction resolution as described above. In [13] terminology, a datum that unifies with its first argument is referred to as the target, one that unifies with its second as the fix-culprit, and the third as the fix-elective:

```
(Fix-via-alternative-course    ;;rule 7
  (fix (failed-semester ?sem ?prof
                        ?course
                        ?p-cycle
                        ?c-cycle)
       (chosen-course ?prof ?course
                      ?p-cycle ?c-cycle)
       (exchange ?alt-course ?course
                 ?prof
                 ?sem ?p-cycle ?c-
                 cycle))
  ←
  (alternative-course
    ?prof ?course ?alternatives)
  (element ?alt-course ?alternatives))
```

If an exchange is successful in resolving a contradiction, the following rule makes the alternative course a desired course, which in turn causes the first rule to make it chosen, thereby repeating the cycle.

```
(Try-An-Alternative    ;;rule 8
  (exchange ?alt-course ?course ?prof
            ?sem ?p-cycle ?c-cycle)
  →
  ;;mark the alternative course as de-
  sired
  ;;(this will cause the
              Select-Course--Rule to fire
  (desired-course ?prof ?alt-course)
           ;;reject the original course.
  (rejected-course ?prof ?course
                   ?p-cycle ?c-cycle))
```

**An Example**

In order to illustrate how the rules above work, let us consider a scenario where among other data, the following are in the database:

Sartre wants to teach phl304 in fall:
(desired-course sartre phl304)

Sartre has as alternatives to phl304 in the fall, phl310 and phl318:
(alternative-course sartre phl304 fall (phl310 phl318))

Suppose that the maximum allowable sections of phl304 have already been allocated for fall:
(semester-max-constraint-satisfied phl304 fall)

Suppose Sartre is now up for consideration (Anselm has already been considered for phl304):
(assign-courses-to-prof Sartre fall (Anselm) ( ))

The rules try assigning phl304 to Sartre in the fall and fail because the maximum constraint has been satisfied for the term. This creates a contradiction as shown in Figure 3. For simplicity we have truncated the data, assuming that they refer to Sartre in the fall term.

In the dependency network in Figure 3, the contradiction is IN. The TMS attempts to make the contradiction OUT by finding a culprit and invalidating one of

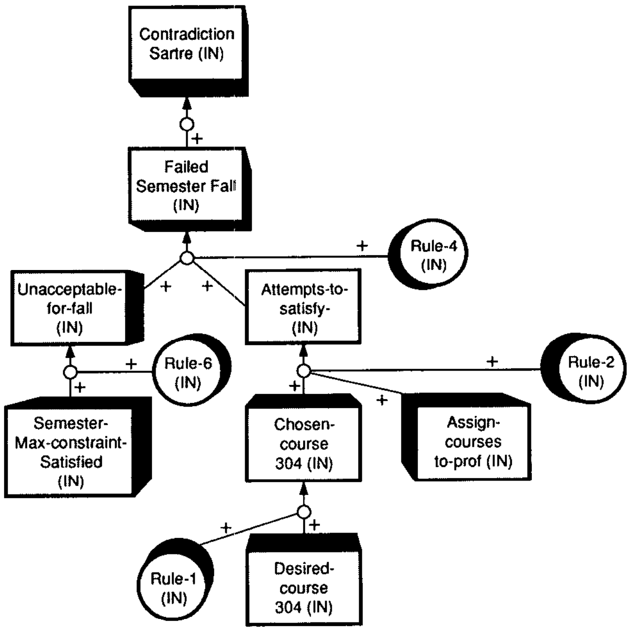


**FIGURE 3. Contradiction State**

its in-supporters. In this case, the culprit is "(chosen-course 304)" (it unifies with the fix-culprit in rule 7) and its in-supporter" (desired-course 304)" is invalidated by putting the "(exchange 310 304)" datum on its outlist, and "(desired-course 310)" gets a valid justification by the exchange datum (via rule 8) on its inlist. Finally, the contradiction is put on the outlist of exchange which makes it (the contradiction) OUT. Part of this new stable and well-founded state is shown in Figure 4.

In summary, the dependency network maintains the reasons for assignments. This includes desired as well as unexpected assignments such as the one resulting from the exchange effected by rule 8. Whenever a justification for some datum becomes invalid, the TMS computes what beliefs must be revised in order to restore consistency and well-foundedness. It is important to recognize that the problem of contradiction resolution is often an underconstrained one, that is, there are many possible labelings that satisfy consistency and well-foundedness. Rules 7 and 8 provide one way of doing it using domain-specific knowledge.

## THE IP ARCHITECTURE

The integer programming formulation consists of about 700 binary variables and 300 constraints. The model has been implemented using the ZOOM (Zero One Optimization Model) library of the XMP package [10]. XMP includes a modeling language, XML, for expressing the problem.

ZOOM solves an integer program as follows. The LP relaxation (ignoring integrality) is first solved. After an LP solution is obtained, a heuristic procedure, called Pivot and Complement (Balas and Martin [1]) attempts to find an integer solution. Basically, this involves a sequence of pivot operations which put all slacks into the basis at minimum cost. If a feasible integer solution is found, it is then improved by flipping variables to their opposite bounds.

As shown in Figure 5, we have implemented a preprocessor that translates the input data used by our expert system into the ZOOM modeling language, XML. The XML interpreter generates an MPS file that is used by ZOOM. The preprocessor takes as input problem data expressed in terms of the object hierarchy of Figure 1. Expressing the constraints in XML requires the preprocessor to translate them into algebraic expressions stated in terms of decision variables. For example, a constraint such as "A faculty member can teach at most 1 graduate course per year" involves searching the ISA hierarchy to locate all instances of faculty, graduate courses and terms, defining the decision variables, and writing out the constraint. In this way, only decision variables essential to the formulation are defined. The number of decision variables can be further reduced by analyzing each teacher's course repertoire and excluding variables corresponding to impossible assignments. For example, if Frege's repertoire slot does not include phl381, he can never be assigned this course, and there is no point in defining a decision variable for this assignment.

The results produced by ZOOM are translated into a schedule for the user. Although not implemented by us, it is also possible to produce other summaries and answer questions using the generated schedule and the object hierarchy. For example answers to questions like "how many faculty are teaching undergraduate courses that meet twice a week," can be very useful to the decision maker.

The preprocessor and translator are both implemented in Common Lisp. The experiments were carried out on a SUN-3 workstation, which is also one of the platforms on which the expert system has been developed.

## ANALYSIS OF RESULTS

The data used by the ES and IP models was provided by the expert. It consisted of profiles of his departmental faculty, their teaching preferences, and departmental course requirements over the last two years. The expressiveness and performance of the two models could therefore be compared with the real plans that had been formulated by the expert and his assistants based on actual data. In this section we present an analysis of the results produced by the IP model relative to the expert model on the available data.

### Performance

Like many integer programs, we found the solution time of the IP model to be highly unpredictable, varying from a few minutes to a few days. In particular, it was extremely sensitive to the teaching load (TL) constraint (number 3 in the IP formulation, the knapsack

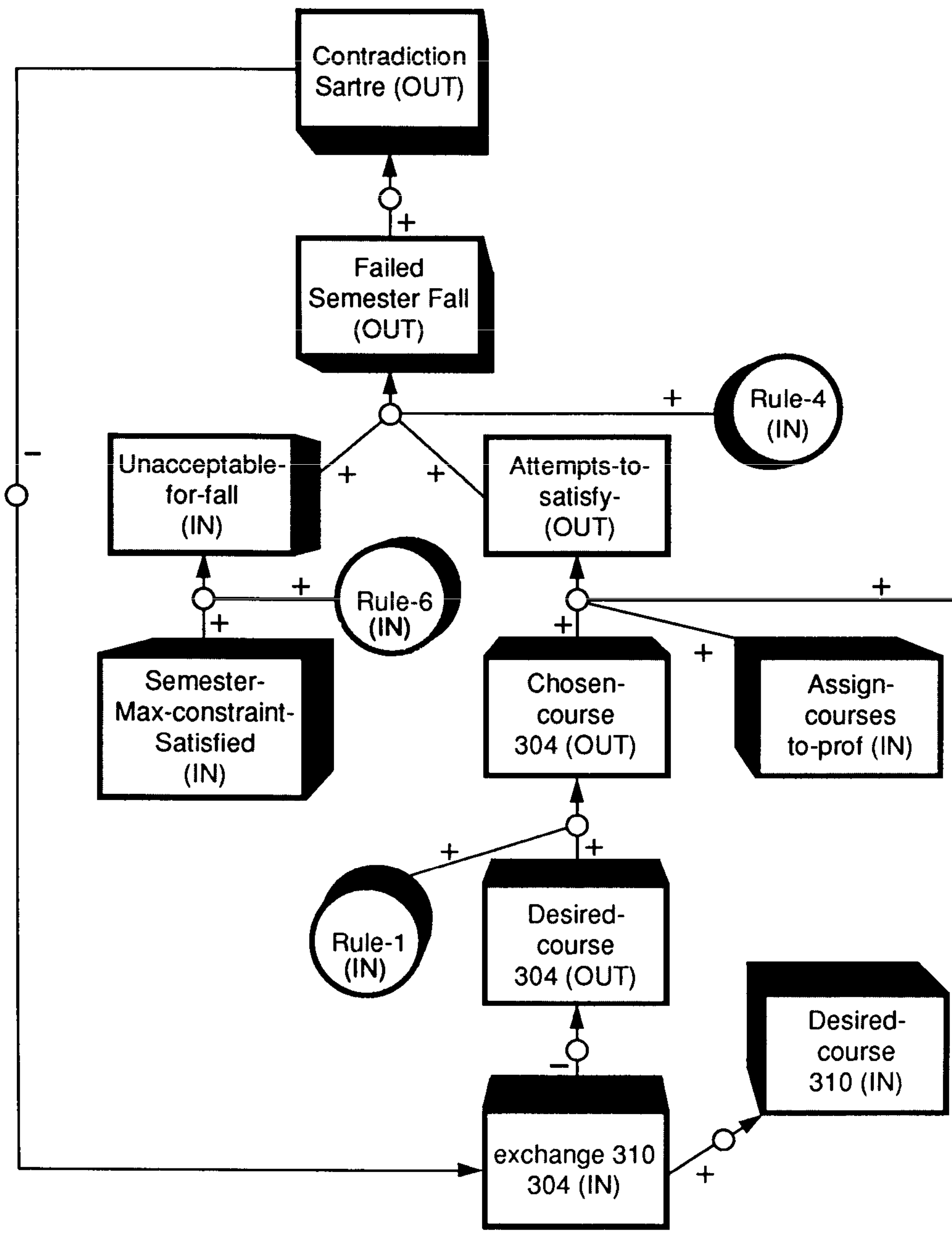


**FIGURE 4. Contradiction-Free State**

constraint). Specifically, varying the value of $T_i$ in the range of interest (i.e. the range in which solutions are feasible or "almost feasible") by as little as 2 percent resulted in orders of magnitude variations in solution time. This was indicative of the tightly constrained nature of the problem—in this case, the bounds on the numbers of sections of each course (constraint 1) were such that it was difficult to satisfy that constraint and the minimum teaching local requirements simultaneously.

In all cases, when a solution was found, it was obtained quickly by the pivot and complement heuristic [1] which is incorporated in the ZOOM code. It always took under five minutes on a dedicated SUN-3. If a feasible solution was not found with this heuristic, it was not found at all.[1] Further, the branch and bound algorithm, which ZOOM resorts to if the heuristic fails, *never* found a solution even after many hours of running time. This is because the number of fractional

[1] The heuristic procedure never found a solution once it started executing its rounding procedure. (The reader interested in the description of these heuristics is referred to [1] and [10].

valued variables remaining after the LP solution, typically between 300 and 500, gives rise to an enormous search tree.

In contrast, the expert system's solution time was less volatile. Under identical conditions on the SUN-3 it usually takes between 1 and 2 hours for it to generate a solution if one exists. If it cannot find one, it generates a partial solution along with a history (in the form of the dependency network) which can be examined to determine why a complete solution could not be found. This is an important advantage of the expert system, which we return to later.

**Control of Reasoning**

In general, the integer program generates a plan that has about three-quarters of the same assignments as those made by the expert. In some cases the expert was pleasantly surprised by its decisions, but in a larger proportion of the cases the differing assignments were judged to be undesirable. The most common manifestations of such undesirable results were cases where teachers had too many preparations, too few preparations, and an unnecessarily heavy load of students (e.g., two huge classes that were not required for the TLs).

Based on the expert's analysis of the solutions generated by the IP, we have identified three reasons for the differing assignments which we call single objective limitations, compiled knowledge limitations, and global optimization limitations.

**Single Objective Limitations**

The objective function used in the formulation expresses one goal of the decision maker: to give all teachers their desired courses to the extent possible. However, in reality there are other goals that the expert tries to satisfy simultaneously. One of these is to ensure that as far as possible, each teacher's load is as close as possible to the minimum required. What is required, therefore, is a multi-objective formulation. Unfortunately, this makes solution extremely difficult. Even if the problem is solvable, a "frontier" of optimal solutions is generated which must be evaluated by the decision maker. For large problems involving discrete choices, analyzing the goodness of alternative sets of assignments can be difficult.

While the single objective formulation works well in most cases, it sometimes produces undesirable results. For example, consider two teachers, Helga and Hegel, who have been assigned courses that result in TLs of 9 and 12 respectively and that 9 is the minimal requirement. Assume that Hegel is more flexible than Helga. Now, if a section of Ethics needs to be taught and Helga and Hegel are the only qualified teachers, Hegel will be assigned since his cost coefficient is lower than Helga's. Clearly, if balancing loads is a concern, the course should be assigned to Helga.

It is possible to alleviate this problem to some extent by expressing the objective function as a constraint, but this too has severe limitations. One must decide on a reasonable TL upper bounds for each teacher. In this example, if Hegel's upper bound is 14, the course will be assigned to Helga, as desired. However, setting such bounds is difficult. If an inappropriate value is used, feasible solutions can be excluded.

In contrast, it is relatively easy to encode knowledge about multiple objectives in the expert system rules. For example, in the Hegel/Helga case, it is possible to have in the antecedent of a rule that attempts to assign courses (such as the select-course rule) a form such as "(least-loaded ?prof))" which would result in the rule condition becoming true (unification succeeding) only with the least-loaded professor. Such rules ensure that "locally good" decisions will be made but they do not guarantee a global optimal solution. We shall elaborate on this point in the following section.

**Compiled Knowledge Limitations**

The behavior of the system and the solution are very sensitive to the cost coefficients. Each teacher has three cost values, corresponding to the penalty associated with assigning a desired course, an explicit alternative, and an implicit alternative (a course that the teacher is capable of teaching but did not ask for). The values are organized as shown in Table I.

Observing the first column, we see that a teacher with less flexibility (CFI = 1) will get preference for a desired course over a teacher that has more flexibility. Moreover, the penalty associated with giving the former an explicit alternative instead of a desired course (9) is higher than doing it for the latter (1). For conflicting desired courses, if the less flexible teacher gets the desired and the more flexible one gets an explicit alternative, the cost is 1 + 6, otherwise it is 5 + 10. Similar reasoning applies to the situation where there are conflicts in the explicit alternatives. Finally, it should be noted that the coefficients are designed so that a desired plus explicit alternative combination is always preferred to a desired plus implicit alternative combination.

**TABLE I. Organized Course Values**

| | Desired | Explicit alt. | Implicit alt. |
|---|---|---|---|
| CFI = 1 | 1 | 10 | 19 |
| CFI = 2 | 2 | 9 | 18 |
| CFI = 3 | 3 | 8 | 17 |
| CFI = 4 | 4 | 7 | 16 |
| CFI = 5 | 5 | 6 | 15 |

The basic problem with the cost coefficients is that they incorporate a lot of compiled knowledge about preferences, flexibility and trading criteria which makes the behavior of the system somewhat unpredictable. We have found that changing the cost values and differentials can have significant unforeseen (desirable or undesirable) consequences on the assignments in the following ways. When the differentials between explicit and implicit alternatives are made larger, the penalty associated with assigning implicit alternatives is high, hence fewer of these are assigned. However, this also

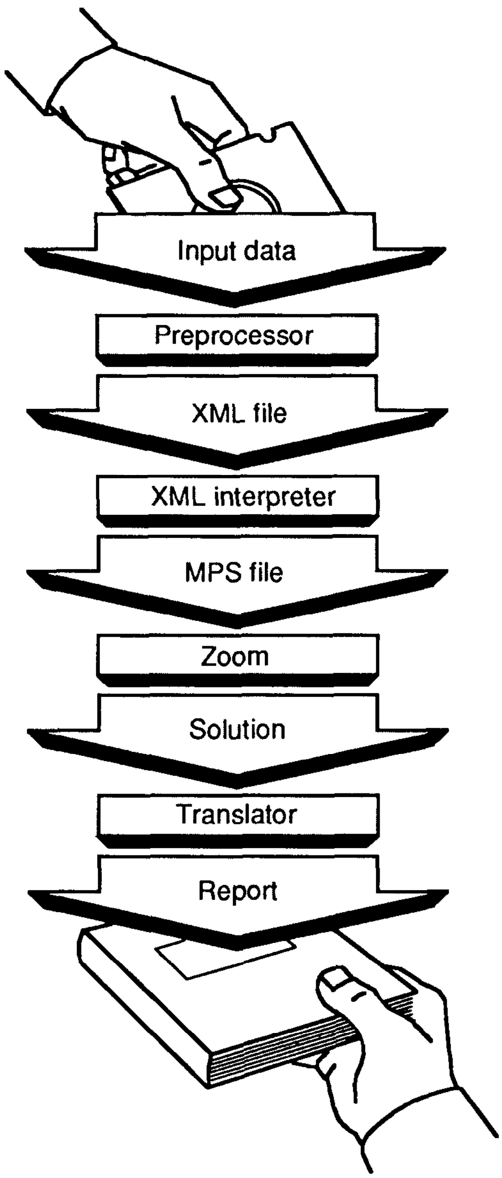


**FIGURE 5. ES to ZOOM Translation**

has the effect of reducing the desired courses assigned since in conflicting situations the penalty associated with assigning explicit alternatives relative to the desired courses is low. When both types of differentials, desired-explicit and explicit-implicit, were increased, there were a few changes in assignments. However, we were not able to determine a general pattern underlying these changes nor the reasons for the changes.

Another related difficulty with compiled knowledge is that of encoding complex preferences in it. For example, it is usually preferable to teach two instead of three courses but if someone needs to teach three (to make the required TLs) it is preferable to have two sections of one course and one section of another instead of one section each of three different courses. This is because there is a diminishing marginal effort associated with teaching an additional section of a course. Such knowledge is not expressed in the coefficients of the objective function, but it is important in matching the expert's behavior.

Another type of knowledge that is difficult to express either in the constraints or the objective function is one involving unusual situations, that is, an action that is rarely undertaken but is a good one under some circumstances. For example, a seminar required in the spring term would not normally be swapped into the fall term except under certain unusual circumstances. Clearly, it is not appropriate to exclude such actions via constraints. Nor is it appropriate to attach heavy penalties to such actions, since this would prevent them from being taken even under appropriate conditions. In the expert system, unusual situations can be left to the contradiction resolution (backtracking) by embedding the appropriate knowledge required for backtracking in fix rules (such as rule 7).

The examples above bring out some of the differences in how the IP and ES models incorporate problem solving knowledge. The ES proceeds locally in a GPS-like manner, reducing the differences between an evolving partial solution and the requirements based on preference information encoded as heuristic rules. In effect, the rules are sensitive to the state of the evolving solution, giving the system builder considerable control over inference.

It is easy to define rules that encode knowledge about the types of preferences discussed earlier. That is, preferences for assigning additional sections of a course before considering a new course, preference in unusual situations, or for assigning courses to less loaded teachers as discussed earlier. In the IP formulation, however, such knowledge is compiled into the coefficients. Given the inevitable loss of information resulting from this transformation it is not surprising that there is a general loss in control over reasoning with the IP model. In other words, in the IP formulation, all such preference knowledge is usually compiled into one global objective function, which controls how the search space is explored. We will now comment on some of the consequences of global optimization.

**Global Optimization Limitations**

While the minimization of cost is designed to maximize the extent to which teachers are assigned desired courses as a whole, the system has a tendency to schedule sections that are as close as possible to the lower bound since this also minimizes cost. This can have the effect of not assigning such courses to teachers that desired them. To illustrate, if at most one section on Ethics should be scheduled, the system has a tendency to schedule none. This turns out to be undesirable in situations where a professor requested Ethics but was assigned an alternative instead.

Essentially, this problem would be avoided if in the case of a professor asking for that course, the lower bound is set to one instead of zero. In effect, the constraint is conditional on the data. However, deciding the appropriate bound based on the input data is tricky since tightening it could rule out feasible solutions or interfere negatively in unforeseen ways with others' assignments.

The only way to express constraints as conditional on the data is to use *non-monotonic justifications*. For the example above, a rule with an "(unless . . . )" form could be used to set the bound. Then, an assignment would have the datum matching the condition in the *unless* in its outlist, making it clear that one section was scheduled since a professor desired it. If that professor were to go on leave, invalidating the desire for that course, the TMS could automatically validate the datum specifying the bound of zero. In general, a TMS is a natural mechanism for modeling default reasoning of this type.

Another consequence of global optimization is the lack of explanation for its decisions. In analyzing the results of the IP we were generally able to infer after analyzing the data in detail, why a teacher had not been assigned a desired course. For the most part, this happened when there was competition for a limited number of sections. Often, however, the rationale for assignments could not be determined even after considerable analysis of the constraints, preferences and flexibility indices. In such cases, global optimization essentially obscures the reasons for assignments.

In contrast, the ES attempts to make good local assignments whose justifications are recorded by the TMS. The justifications are extremely useful for purposes of explanation and for incremental revision of existing decisions in an evolving solution. Although the ES has no notion of a global optimum, the more the knowledge provided to the system for resolving conflicts, making choices, etc., the better the quality of its solution. The factors that determine whether the expert system will work better than the IP model are the extent to which it is important to specify complex preferences of the type described in the preceding discussion, and the ease with which such knowledge can be specified by the expert. These two factors interact in a complex manner. If complex preferences need to be specified, it is usually indicative of the complexity of defining optimal solutions, and hence the limitations of the objective function as the mechanism driving the search. However, in such situations the expert also finds it increasingly difficult to specify the preference knowledge in terms of abstractions. Over time, this can result in a situation where interactions among the various pieces of knowledge become very complex, thereby eroding the modularity of the knowledge base.

### The Need for Partial Solutions

A major problem during the early runs of the IP model was its inability to find a feasible integer solution even after many hours of running time. Two factors contributed to this situation. Firstly, the data were often such that the constraints were not satisfiable. For example, if three sections of a course were required and only two people were qualified to teach it, clearly, no solution would be found. Secondly, we found that the knapsack constraints, on TLs, were often too tight for it to find a feasible integer solution.

We solved the first problem by analyzing the behavior of the expert system. While that system is also often unable to find a feasible solution, it generates a useful partial solution indicating those "holes" in the schedule that still need to be filled.

Partial solutions are extremely useful to the decision maker. There is no guarantee that the skills of the teachers will cover the requirements (particularly if a significant number of faculty are on leave). In such cases, and the decision maker needs to know what the holes are. Typically, there are several holes in the schedule in the initial draft. These are patched by hiring visiting professors, lecturers, or graduate students. In this respect, the integer program is deficient because if it fails to find a feasible integer solution, it runs virtually forever and finally provides no useful information to the decision maker. What is required in such cases is an integer solution which even though not feasible, is *close* to feasible.

Even though the results of the expert system pointed us to the section requirements constraints to loosen, we still did not manage to obtain a solution until we simplified the problem by excluding the TL constraints, solving it as a transportation problem. We then introduced lowered TL requirements. As we mentioned earlier, we found that the ability of the system to find an integer solution was extremely sensitive to the TL value. On analyzing the expert's behavior, we found that he dealt with the TL requirements by first ignoring them, making assignments that put teachers' loads in the "ballpark," and then massaging the schedule to satisfy the requirements. For the most part, this massaging consists of giving teachers larger sections which have higher credits (instead of giving additional courses which increase the number of preparations) or assigning them light administrative responsibilities for which they earn small amounts of credit.

## PLAN REVISION

A major drawback of the IP model is its lack of support in making revisions to a plan whose underlying assumptions often change. Planning course assignments is based on assumptions about enrollments (which determine the numbers of sections planned), and faculty availability. If enrollments turn out to be higher than expected, additional sections must be scheduled. If a faculty member gets a grant or goes on leave, substitute teachers must be found. In all such cases, it is important that the overall plan be perturbed as little as possible.

Several types of actions can be taken by the decision maker when assumptions change. Depending on the change, the action can include hiring assistant instructors, hiring visiting professors, and swapping assigned courses among faculty. Hiring a visitor is usually feasible only if there is sufficient time to negotiate a contract.

Some changes are easy to manage. For example, if a faculty that goes on leave was teaching low level undergraduate courses, instructors can be hired to fill in. However, if graduate courses are involved revision becomes more complex since such courses can be highly specialized, making it difficult to find faculty qualified to teach them. Also, faculty are limited to a maximum of one graduate course per year. In summary, determining who to assign to an unassigned graduate course can be difficult, depending on how many faculty are qualified to teach the course, how flexible they are, whether they are already scheduled to teach a graduate course, and whether substitutes can be found for such courses.

The IP model supports revision as follows. The decision maker must specify what subset of the existing set of assignments can be changed. The complement set is considered fixed; the assignments in this set are therefore added as constraints and the problem is re-solved. In effect, the decision maker must specify what part of the plan is fixed and what is variable. Further, the decision maker must make a judicious choice in specifying what combination of lecturers, Assistant Instructors and visiting professors should be considered to fill the newly created holes in the plan.

Unfortunately, it is usually difficult for the decision maker to specify in advance with any degree of confidence which parts of the plan should be considered variable. Rather, the process of figuring out what to change requires negotiation, the results of which serve as further input in determining what parts of the model can be considered changeable. In effect, *figuring out what should be changed* is where support is most needed. In this sense, the IP model requires the decision maker to do too much. Specifically, if changes are to be minimal, he must keep the variable set small, otherwise the new solution can contain too many changes. However, this small set of changeable assignments should also result in a solution being found, otherwise the exercise is of little use. Determining the appropriate set can require considerable trial and error; finding it is therefore the really difficult part of the problem where support is most needed.

Another aspect of revision is that certain changes actually require that changes be made to the constraints. For example, if an instructor who goes on leave was teaching a course that is not strictly required, it is usually not re-assigned to anyone when the IP model is run with the new data since the objective is to minimize cost. Clearly, this is problematic if that course has already been listed as an offering. In such cases, what is really required is that the constraints be modified to state that the listed course *must* be offered.

In order to support revision, it is necessary that a system actually *suggest* alternative courses of action to the decision maker. For this to be possible, it is necessary for it to record the rationales for existing assignments. For example, if an instructor $I_1$ who goes on leave was teaching $X_1$ which was desired by but not assigned to $I_2$ (perhaps due to an exchange like the one illustrated in the example), it might make sense to consider whether $I_2$ should now teach $X_1$. This type of reasoning is supportable with a truth maintenance system. In a TMS, for example, one of the justifications for the assignment $I_2 \leftarrow X_2$ would be the fact that $I_1$ was assigned $X_1$. When this latter proposition is no longer true, the justification for $I_2 \leftarrow X_2$ becomes invalid, making it possible to assign $X_1$ to $I_2$. This process can be repeated recursively until a consistent set of assignments is found.

In practice, however, it would probably be undesirable for the above process to happen automatically for two reasons. Firstly, it is conceivable that taking away $X_2$ from $I_2$ could create an even bigger problem if it is difficult to find someone to teach $X_2$. Secondly, it would not make sense to take such a course away if a significant amount of preparation has already gone into preparing for that course or if the teacher assigned to it is inflexible.

Regardless of these limitations, a TMS should prove to be useful. Even if it fails in repeated attempts at finding a solution, the reasons for failure can be recorded and presented to the decision maker. In this way, even if the TMS does not find a solution, it provides useful information that can be used to find a solution. Since it is desirable to try several avenues simultaneously, an ATMS [3] might be suitable even though in principle any TMS could be used.

The problem of determining how breakable an assignment is is a much more difficult problem. It depends on the flexibility of the teacher, how suited substitutes are for a course, and how much time has already gone into preparation. We are currently in the process of trying to formalize these concepts so that they might be representable and used by a TMS.

## DIRECTIONS FOR FUTURE WORK

It is clear that the optimization problems solvers such as ZOOM must be made much more flexible if they are to prove useful as decision support tools for practical problems. Based on our experience with the experiments and analysis of expert behavior, we feel that there are at least two directions that are worthwhile pursuing. First, a more expressive interface is required which allows the problem data to be specified as "naturally" as possible. Second, it should be possible to augment ZOOM so that it provides more useful output to the user.

On the interface front, we have found that specifying problem data in terms of the object hierarchy and defining constraints in terms of such objects is a very useful functionality for the user. In effect, we have built a layer on top of XML. Currently languages like XML and GAMS are used directly as specification languages. While they (especially GAMS) allow the user to specify the problem in terms of a compact notation, they are still relatively low-level utilities. We feel there are significant productivity gains possible by providing a higher level modeling environment where a user can specify an arbitrary class hierarchy and constraints over it, and have the mathematical formulation generated automatically. We are currently working on the primitives that such an environment must have for it to be able to work for all mathematical programming problems.

We also feel it should be able to build in a truth maintenance functionality into the optimization package, making it more flexible and useful to the user. In ZOOM, for example, considering that the branch and bound is usually unsuccessful (in our case it was *always* unsuccessful), it makes sense to try and generate an "almost feasible" solution in cases when the Pivot and Complement heuristic fails.

When the heuristic fails, the system knows which constraints are being violated (in our problem these were the knapsack constraints). At this point, if the number of non-integer variables is not too large (i.e. less than about 30), the system could perform a local search around these variables only (keeping values of all integer variables fixed), that is, attempt different combinations of integer values for the non-integer variables, and keeping a record of the contradictions. Such a history could be used by the system (or the user) to decide which constraints to loosen in order to generate a feasible solution. In effect, this boils down to augmenting the Pivot and Complement heuristic with a primitive truth maintenance system. Actually, this is quite similar to what the expert does in overconstrained situations (i.e. ignore TLs, obtain a solution, and gradually introduce the TLs). By enabling the system to loosen constraints in situations where no solution is in sight, it can begin to approximate the problem solving behavior of experts. As a next step in this research we are considering ways of incorporating a truth maintenance functionality into optimizers to handle such situations.

***Acknowledgments.*** We are extremely grateful to Ted Stohr, Charles Petrie and Elaine Rich for comments that have greatly improved this article. The expert system rules were formulated by Charles Petrie, and the system was implemented by Donald Steiner and Petrie. We thank Donald Steiner for providing us with the rules used for the belief revision example in this article. Robert Causey provided extensive critiques of the solutions produced by the integer programming model. Finally, thanks to Roy Marsten for ZOOM.

**REFERENCES**

1. Balas, E., and Martin, C.H. Pivot and complement—A heuristic for 0-1 programming. *Management Sci. 26*, 1 (Jan. 1980), 86–96.
2. Dantzig, G. *Linear Programming and Extensions.* Princeton Univ. Press, 1963.
3. de Kleer, J. An assumption-based TMS. *Artif. Intell. 28*, 2 (March 1986).
4. Dhar, V. A truth maintenance system for supporting constraint-based reasoning. *Decis. Support Syst.* (Fall 1989).
5. Doyle, J. A truth maintenance system. *Artif. Intell. 12*, 3 (1979).
6. Garey, M., and Johnson, D. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Co., N.Y., 1979.
7. Hinton, G.E. Relaxation and its role in vision. Ph.D. thesis, Univ. of Edinburgh, 1977.
8. Hooker, J.N. A quantitative approach to logical inference. *Decis. Support Systems 4*, 1 (March 1988).
9. Marr, D. *Vision.* W.H. Freeman and Co., N.Y., 1982.
10. Marsten, R. ZOOM/XMP User's Manual. Release 4.0, XMP Optimization Software Company, Tucson, Ariz., July 1987.
11. McAllester, D. Reasoning Utility Package. MIT-AI Lab Memo 667, April 1982.
12. Petrie, C., Russinoff, D., and Steiner, D. Proteus 2: System Description. MCC Technical Report AI-136-87, May 1987.
13. Petrie, C., Revised dependency-directed reasoning for default reasoning. *Proceedings of the Sixth National Conference on Artificial Intelligence* (AAAI-87), July 1987, pp. 167–172.
14. Petrie, C., Causey, R., Steiner, D., and Dhar, V. A Planning Problem: Revisable Academic Course Scheduling. MCC Technical Report ACT-AI-020, June 1989.
15. Proteus. Proteus: A Default Reasoning Perspective. MCC Technical Report AI-352-86, 1986.
16. Reinfrank, M. Lecture Notes on Reason Maintenance Systems. Technical Report INF2 ARM-5-88, Siemens AG, Munich, W. Germany, 1988.
17. Stefik, M. Planning With Constraints. Ph.D. thesis and Technical Report STAN-CS-80-784, Department of Computer Science, Stanford Univ., Stanford, Calif., 1980.
18. Waltz, D. Understanding Line Drawings of Scenes With Shadows. *The Psychology of Computer Vision*, P.H. Winston, Ed. McGraw-Hill, N.Y., 1975.

ABOUT THE AUTHORS:

**VASANT DHAR** is associate professor of Information Systems at the Leonard N. School of Business, New York University. His research focuses on the structure of planning and design problems and the development of representation and reasoning formalisms for knowledge-based systems dealing with such problems.

**NICKY RANGANATHAN** is a Ph.D. student of Information Systems at the Leonard N. School of Business, NYU. He is also affiliated with MCC and the Eastman Kodak Corporation where he is analyzing the process of design with the objective of identifying what parts of this process can be supported with intelligent design systems, and the representations needed to build such systems.

Authors' Present Address: Information Systems Dept., New York University, 40 W. 4 St., Room 619, New York, NY 10003.