KNOWLEDGE REPRESENTATION FOR SCHEDULING CONSTRUCTIONS

M. A. Pollatschek
Technion

ABSTRACT

Knowledge may be conveniently represented in Expert Systems for scheduling constructions by means of frames, or equivalently, by assemblies of feature sets (pairs of the type NAME_OF_FEATURE, VALUE_OF_FEATURE). Each frame is a description of a constraint which is to be observed in a feasible schedule. Such a representation is simple for the user but tends to take sizable memory and extraction of information from it by pattern-matching is time-consuming as well. We define the property of decomposability of the feature sets. It may not be present in all applications, but it deffinetely exists in the case of construction scheduling. Utilizing decomposability leads to a very sizable reduction of the knowledge base and opens a possibility of efficient scanning of it. In this paper we define decomposabilty and discuss its application to feature sets. We also show how to compile the feature sets arising in constraction scheduling so as to lead to extremely compact rules and efficient scanning.

1. INTRODUCTION.

The CPM and PERT methods for scheduling in construction has been well established for more than 20 years by now. They provided good answers for revealing bottlenecks and are subject to investigations still today (see e. g. [4]). However, since the introduction of these techniques two important developments has arisen, namely the emergence of personal computers and expert systems. Today we expect the scheduling software to do more than just CPM or related tasks. It should also be somewhat intelligent: it should check the human for plausible input and create possible solutions, while all this should run on a personal computer in reasonable time. There are research efforts in this direction (see e. g. [5]), which show how to make computers intelligent by building a suitable knowledge base. A knowledge base is the data which describes rules. If a schedule satisfies all the rules then it is acceptable or feasible. However, there is no approach known to me which fully exploit the specific situation in construction.

What we want to propose in this paper is a knowledge base for construction scheduling which is compact enough to be successfully employed from a personal computer, and at the same time powerful like any professional expert system shell, which utilizes the specific features characterizing our domain.

The special in our case is that we always consider a specific job (like excavation) occurring in a time period (say in between 4 to 7 of February) and demanding given equipments and manpower.

Note that we have here a triad of the elements or atoms: job, time, resource. We consider discrete time periods like days. It is clear that the other elements are naturally discrete. We denote the triad so :

[j, t, r]

where j is in the set {1..J}, t is in the set {1..T} and r is in the set {1..R}, assuming J kinds of jobs, T time-slots and R sorts of resources while we assign natural numbers to the specific elements.

Any schedule is an assembly of triads. However, not every assembly is acceptable or feasible. For example we cannot start building before excavation or we have to invest a given amount of working days in a given job. The rules for inclusion or exclusion of triads in a feasible schedule consist of the knowledge base. We take advantage of the fact that a rule can be treated as a constraint on the triads which may be in a feasible schedule. A few examples will be given in the following sections.

We shall show that the rules can be represented in frames of only two kinds, provided that a property which we shall define, is satisfied. We claim that our problem can always be formulated so that this property holds. The main result is that these frames can be held very compactly in the random access memory of any personal computer and these compact forms can be very efficiently scanned to check a particular constraint.

2. KNOWLEDGE BASE IN FRAMES.

The term "frame" was coined by Minsky [6], and in the Handbook of AI [1] is described as "declerative and procedural information

in predifined internal relations". For our present purposes it can be imagined as an assembly of diads, where the first element is the name of the feature and second is its value (in general, frames can carry much richer information - see for example [2]). The diads are sometimes called feature sets.

For example, consider the fact that no crane (say resource # 5) can participate at any time in more than in one task. This can be represnted in the following frame :

JOB_SET	: {1,..,J}
CONSIDER_MEMBERS_OF_JOB_SET	: TOGETHER
TIME_SET	: {1,..,T}
CONSIDER_MEMBERS_OF_TIME_SET	: SEPARATELY
RESOURCE_SET	: {5}
CONSIDER_MEMBRS_OF_RESOURCE_SET	: NO_MATTER
TYPE_OF_CONSTRAINT	: LESS_THAN_OR_EQUAL
CONSTRAINT_CONSTANT	: 1

The _SET feature refers to the atoms participating in the constraint, while the CONSIDER_ feature defines the mode of participation. The former has the appropriate set as the value part and the value of the latter can be either TOGETHER or SEPARATELY or NO_MATTER. SEPARATELY means that for each member in the set (in this example for each time-slot) we have separate constraint, TOGETHER signifies that we take the members of the set together in a single constraint. When the set has only one member, TOGETHER and SEPARATELY implies the same, so we can declare it as NO_MATTER. Thus the above frame states that for each t in {1,..T}, from all the J triads:

[1, t, 5] , [2, t, 5] ,.., [J, t, 5]

only one or less may appear in any feasible schedule, or

equivalently, that resource # 5 (= crane) at any t (time-slot) can do at most one job.

Another example: in the first 3 days (= time-slots) equipment (= resource) # 8 and # 11 is not available. Here is the frame describing the situation :

JOB_SET	: {1,..,J}
CONSIDER_MEMBERS_OF_JOB_SET	: SEPARATELY
TIME_SET	: {1,2,3}
CONSIDER_MEMBERS_OF_TIME_SET	: SEPARATELY
RESOURCE_SET	: {8,11}
CONSIDER_MEMBRS_OF_RESOURCE_SET	: SEPARATELY
TYPE_OF_CONSTRAINT	: EQUAL
CONSTRAINT_CONSTANT	: 0

Here we add a further convention (not showing up in our previous example), namely that if value SEPARATELY turns up more than once we take the Cartesian product of all the sets considered SEPARATELY. Thus the number of individual constraints here is

$$J \times 3 \times 2,$$

and they are : for each j in {1,..,J} and each t in {1,2,3} and each r in {8,11} the number of appearance of the triad [j,t,r] is zero, thus cannot be in any feasible schedule. In the same fashion, if we have more than once TOGETHER in the frame, triads formed from their Cartesian product will take part in a single constraint. Therefore, the frame :

JOB_SET	: {3,4}
CONSIDER_MEMBERS_OF_JOB_SET	: TOGETHER
TIME_SET	: {5,6}
CONSIDER_MEMBERS_OF_TIME_SET	: TOGETHER
RESOURCE_SET	: {7,8}
CONSIDER_MEMBRS_OF_RESOURCE_SET	: SEPARATELY
TYPE_OF_CONSTRAINT	: EQUAL
CONSTRAINT_CONSTANT	: 1

indicates that exactly one of the four triads below is to be in any feasible schedule :

[3, 5, 7] , [3, 6, 7] , [4, 5, 7] , [4, 6, 7],

as well as from the following four ones :

[3, 5, 8] , [3, 6, 8] , [4, 5, 8] , [4, 6, 8].

To sum up, we can list the triads of constraints we considered up to now with the following convention: form the Cartesian product of all the sets with TOGETHER feature value, and let this set be denoted by X. Denote by Y the Certesian product of the sets with SEPARATELY feature value. Then we have | Y | constraints. (|.| stands for the cardinality of the concerned set, or its number of members. If the set is empty |.| is defined as one - not zero as is usual in mathematics.) In each constraint we have | X | triads, while each triad is a member in the Cartesian product X x Y. The question is whether the triad set of any constraint appearing in construction scheduling can be obtained in such a way. This will be discussed in the following section.

3. THE PROPERTY OF DECOMPOSABILITY AND IMPLICATIVE CONSTRAINTS.

A very small example demonstrates that the way we constructed the triad sets does not always work. Consider :

[1, 2, 3] [4, 5, 6].

It is clearly impossible to write this set as a Cartesian product of JOB_SET, TIME_SET and RESOURCE_SET. Hence, it is necessary to define the property which enables the construction of the last section, which we refer to as the property of decomposability.

A triad set has the property of decomposability if it can be written as a Cartesian product of three sets, one containing the jobs, one - the times and one - the resources. The operation of the actual decomposation into job, time and resource sets will be termed factorization, while the three sets will be called the factors.

We have seen that there are instances having this property, but the frames we have studied meant essentially the following : a given number of triads from the set should or should not be present in any feasible schedule. Not all the constraints are of this type, which will be referred as the enumerative kind. Let us think about a job (say # 7) which can start at any time but needs three resources (say # 1,2,3). We can formulate it as follows :

[7, t, 1] in schedule => [7, t, 2] in schedule for each t=1..T
[7, t, 1] in schedule => [7, t, 3] in schedule for each t=1..T

Here => means implication. We shall also say that the presence of

[7, t, 1]

(or satisfying a constraint) triggered another constraint. Thus, we need constraints describing implications when the parameter is in some TIME_SET. Basing the formulation on the examples of the previous section, we shall picture an implication as one between two enumerative constraints. The only link between the two are the time parameter or the constraint parameter (or both), which are interpreted in the following meaning. Whatever values of the time parameter, t or the constraint parameter, c take in one triad set, they must take the same value in the other one as well. However, we allow a constant displacement, d in the time parameter of the second constraint. Thus, [., t, .] in the first has counterpart [., t+d, .] in the second constraint. This interpretation means that CONSIDER_MEMBERS_OF_TIME_SET's value is SEPARATELY in implications.

The general form of the implicative constraint frame is as follows :

JOB_SET_1
CONSIDER_MEMBERS_OF_JOB_SET_1
TIME_SET_1
CONSIDER_MEMBERS_OF_TIME_SET_1
RESOURCE_SET_1
CONSIDER_MEMBRS_OF_RESOURCE_SET_1
TYPE_OF_CONSTRAINT_1
CONSTRAINT_CONSTANT_1
JOB_SET_2
CONSIDER_MEMBERS_OF_JOB_SET_2
TIME_SET_2
CONSIDER_MEMBERS_OF_TIME_SET_2
RESOURCE_SET_2
CONSIDER_MEMBRS_OF_RESOURCE_SET_2
TYPE_OF_CONSTRAINT_2
CONSTRAINT_CONSTANT_2
TIME_DISPLACEMENT
CONSTRAINT_PARAMETER_RANGE

Here enumerative constraint number 1 (the implicator) implies number 2 (the implicand) and the CONSTRAINT_CONSTANT's value may be besides a constant in the usual mathematical meaning also a function of a single parameter, c in the CONSTANT_PARAMETER_RANGE. For example, in the case of three jobs needing three resources at one time, the first implication can be written as the following frame:

```
JOB_SET_1                          :  {7}
CONSIDER_MEMBERS_OF_JOB_SET_1      :  NO_MATTER
TIME_SET_1                         :  {1..T}
CONSIDER_MEMBERS_OF_TIME_SET_1     :  SEPARATELY
RESOURCE_SET_1                     :  {1}
CONSIDER_MEMBRS_OF_RESOURCE_SET_1  :  NO_MATTER
TYPE_OF_CONSTRAINT_1               :  EQUAL
CONSTRAINT_CONSTANT_1              :  c
JOB_SET_2                          :  {7}
CONSIDER_MEMBERS_OF_JOB_SET_2      :  NO_MATTER
TIME_SET_2                         :  {1..T}
CONSIDER_MEMBERS_OF_TIME_SET_2     :  SEPARATELY
RESOURCE_SET_2                     :  {2}
CONSIDER_MEMBRS_OF_RESOURCE_SET_2  :  NO_MATTER
TYPE_OF_CONSTRAINT_2               :  GRATER_THAN_OR_EQUAL
CONSTRAINT_CONSTANT_2              :  c
TIME_DISPLACEMENT                  :  0
CONSTRAINT_PARAMETER_RANGE         :  (0,1)
```

The frame means that the presence (if c=1) of resource # 1 with job # 7 requires the presence of resource # 2 as well, but not necessarily conversely.

We claim that any construction scheduling knowledge base can be described by the two kinds of frames discussed above. Unfortunately, we do not have the space to demonstrate it with examples, but the reader can hopefully convince himself or herself that our claim holds. In case of difficulty please, contact the writer. The advantage of two fixed form of frames is not only the parsimony itself, but rather as being the starting point of further possibilities, that are impossible to attain with general frames. They are the topic of the remaining sections.

4. CONSISTENCY AND REDUCTION OF ENUMERATIVE FRAMES.

In real life applications the number of frames can be numerous and their consistency are to be checked, i.e. to ascertain that no two frames are in contradiction. It is also possible that one frame implies the other or two frames can be combined to one; in this case we can reduce the knowledge base. We shall discuss here checking two frames for these reasons. (We shall not attempt to check contradictions or implications resulting from more than two frames.) This and the following section introduce the tests for this purpose.

Clearly, two frames can be compared only if they are of the same kind: either both enumerative or both implicative. First consider enumerative frames. Denote by X(i) the Cartesian product of sets with TOGETHER value, and by Y(i) those with SEPARATELY value for frames # (i=) 1 and 2. For the discussion for the rest of the paper NO_MATTER will be given the SEPARATELY value.

Consider the following example of two frames :

```
JOB_SET                              : {1,2}
CONSIDER_MEMBERS_OF_JOB_SET          : SEPARATELY
TIME_SET                             : {1,..,T}
CONSIDER_MEMBERS_OF_TIME_SET         : SEPARATELY
RESOURCE_SET                         : {5}
CONSIDER_MEMBRS_OF_RESOURCE_SET      : NO_MATTER
TYPE_OF_CONSTRAINT                   : LESS_THAN_OR_EQUAL
CONSTRAINT_CONSTANT                  : 1

JOB_SET                              : {7,9}
CONSIDER_MEMBERS_OF_JOB_SET          : SEPARATELY
TIME_SET                             : {1,..,T}
CONSIDER_MEMBERS_OF_TIME_SET         : SEPARATELY
RESOURCE_SET                         : {5}
CONSIDER_MEMBRS_OF_RESOURCE_SET      : NO_MATTER
TYPE_OF_CONSTRAINT                   : LESS_THAN_OR_EQUAL
CONSTRAINT_CONSTANT                  : 1
```

Here X(1) and X(2) are both empty (NO_MATTER is considered always as SEPARATELY), Y(1) and Y(2) are the Cartesian product of all the three sets in each frame and the constraints are the same in both. Note that the following single frame represents the above two :

```
JOB_SET                          : {1,2}
CONSIDER_MEMBERS_OF_JOB_SET      : SEPARATELY
TIME_SET                         : {6}
CONSIDER_MEMBERS_OF_TIME_SET     : NO_MATTER
RESOURCE_SET                     : {1,2,3,4,5}
CONSIDER_MEMBRS_OF_RESOURCE_SET  : TOGETHER
TYPE_OF_CONSTRAINT               : LESS_THAN
CONSTRAINT_CONSTANT              : 3

JOB_SET                          : {2,5}
CONSIDER_MEMBERS_OF_JOB_SET      : SEPARATELY
TIME_SET                         : {1,2,3,4,5,6}
CONSIDER_MEMBERS_OF_TIME_SET     : SEPARATELY
RESOURCE_SET                     : {2,3,4,5}
CONSIDER_MEMBRS_OF_RESOURCE_SET  : TOGETHER
TYPE_OF_CONSTRAINT               : EQUAL
CONSTRAINT_CONSTANT              : 4
```

since the second requires among other things that all the four triads :

[2, 6, 2], [2, 6, 3], [2, 6, 4], [2, 6, 5]

are in a feasible schedule while the first frame implies that at most two only may appear.

This can be generalised by:

Test [2] Two frames contradict each other if the intersection of their Y sets is not empty and fall into one of the four cases in the following table (where TYPE = TYPE_OF_CONSTRAINT, CONSTANT = CONSTRAINT_CONSTANT, EQ = equal, GE = greater or equal, LE = less than or equal). Strictly greater or less type of constraints are not discussed because they can be rewritten as GE or LE types since the sets are discrete. The indicated set inclusions are not strict, they also hold if the sets are equal.

TYPE(1)	TYPE(2)	X(1)...	CONSTANT(1) is ...
EQ	LE	includes ...	greater than ...
GE	LE	includes ...	greater than ...
EQ	GE	includes ...	less than ...
EQ	EQ	is equal to ...	not equal to ...
		X(2)	CONSTANT(2)

Finally, compare the following two frames :

```
JOB_SET                          : {2,5}
CONSIDER_MEMBERS_OF_JOB_SET      : SEPARATELY
TIME_SET                         : {1,2,3,4,5,6}
CONSIDER_MEMBERS_OF_TIME_SET     : SEPARATELY
RESOURCE_SET                     : {1,2,3,4,5,6}
CONSIDER_MEMBRS_OF_RESOURCE_SET  : TOGETHER
TYPE_OF_CONSTRAINT               : LESS_THAN_OR_EQUAL
CONSTRAINT_CONSTANT              : 2

JOB_SET                          : {2}
CONSIDER_MEMBERS_OF_JOB_SET      : SEPARATELY
TIME_SET                         : {1,3,5,6}
CONSIDER_MEMBERS_OF_TIME_SET     : SEPARATELY
RESOURCE_SET                     : {2,3,4,5}
CONSIDER_MEMBRS_OF_RESOURCE_SET  : TOGETHER
TYPE_OF_CONSTRAINT               : LESS_THAN_OR_EQUAL
CONSTRAINT_CONSTANT              : 3
```

Here Y of the first includes Y of the second and about the X set the opposite holds. A typical constraint of the second is that at most three of the following triad set can appear in any feasible schedule

[2, t, 2], [2, t, 3], [2, t, 4], [2, t, 5]

for t=1,3,5,6, while from the first frame it is known that at most two triad can turn up from the set

[2, t, 1],[2, t, 2],[2, t, 3],[2, t, 4],[2, t, 5],[2, t, 6]

for t=1,2,3,4,5,6. It is obvious that the first frame implies the second.

This can be extended by:

Test [3] Frame # 1 is implied by (or follows from) frame # 2 (and therefore may be deleted) if Y(1) is included in Y(2) and fall into one of the five cases in the following table. (Notations and conventions of the previous table apply also here.)

TYPE(1)	TYPE(2)	X(1)...	CONSTANT(1) is ...
LE	LE	is included in ...	GE to ...
GE	GE	includes ...	LE to ...
EQ	EQ	includes ...	EQ to ...
LE	EQ	includes ...	GE to ...
GE	EQ	includes ...	LE to ...
		X(2)	CONSTANT(2)

5. CONSISTENCY AND REDUCTION OF IMPLICATIVE FRAMES.

Now consider the implicative frames. Here we have a pair X and Y sets in each frame, so denote by X_1(i) and Y_1(i) of the sets in the implacator part and by X_2(i) and Y_2(i) of those in the implicand part, where i=1 or 2 stands for the frame. The analogous cases are based on the results of the previous section.

We shall examine this kind of frames without examples. At first we shall discuss when combination is possible. It may be shown that the following holds:

Test [4] If in two frames all the corresponding values concerning the _CONSTRAINT values are the same as well as X_1(1) = X_1(2), X_2(1) = X_2(2), Y_k(1) = Y_k(2) (where k = 1 or 2) and the relation between Y_m(1) and Y_m(2) is as under [1] (where m = 3 - k) then they may be combined, while the m-part of the combined frame is formed as in [1].

Test [5] If the implicator part of frame # 1 follows from the implicator part of frame # 2 for some t and c in the sense of [3] and the implicand parts are in contradiction for the same values of t and c in the sense of [2], then frames # 1 and # 2 themselves are in contradiction as well.

Test [6] If the implicator part of Frame # 1 follows from the implicator part of frame # 2 in the sense of [3] for each c and the implicand part of frame # 2 follows from the implicand part of frame # 1 for each c in the sense of [3] as well, then frame # 1 may be deleted since it is implied by frame # 2.

The successive application of the six tests establishes the consistency of the knowledge base as well as can reduce it to the most compact size. However, even the reduced knowledge base can be of considerable proportions. In the next section we shall see that more drastic reduction is possible.

5. COMPRESSING THE FRAMES.

The two predefined frames allow us to represent them in a much more compact fashion than in conventional general purpose knowledge bases. The idea is to carry only the factors of the triad sets. Since the factors are simple sets, they can be represented in the memory as bit-strings, when 0 at position i stands for the fact that member # i is not in the set, while 1 at position i signifies belonging. (This is the usual compilation procedure for sets in the Pascal language, see [3].)

The space needed for a frame can be calculated from the following table, supposing that J, T, R are each not over 255 (maximum set size in most Pascal implementations) and that the factors are represented as sets :

Feature	Possible values	Number of bits for each feature
Frame-type	Enumerative, implicative	1
_SET	Set	J or T or R
CONSIDER_	SEPARATELY,TOGETHER	1
TYPE_OF_ CONSTRAINT	LE,GE,EQ	2
CONSTRAINT_ CONSTANT	Integer in enumerative String in implicative	8 varying
TIME_ DISPLACEMENT	byte	1
PARAMETER_ RANGE	2 integers	2x8

A few notes are here in order. NO_MATTER will be treated as SEPARATELY, as previously, LE, GE, EQ stand for less than or equal, greater than or equal and equal, respectively as was used above in. As we observed there, strict inequalities can be

rewritten as LE or GE.

For 200 jobs, 100 days and 50 resources we need 362 bits in a numerative and 740 bits (or somewhat more if the strings are longer than one byte) in implicative constraint. This is 46 and 93 bytes, respectively. For comparison, only the feature names take 166 and 377 bytes respectively, so we have an advantage over a lisp-machine or similar workstation for an AI undertaking, based on an interpreter, but the real gain is in the speed of table-lookup. Before turning to this topic let finish with the present example, by assuming 300 rules, half of which is implicative. The memory requirement of such an application is below 21 K bytes!

Now let us review the scanning of the knowledge base as defined above. The basic problem is constraint checking, i.e., to ascertain that a given triad does not violate a rule or triggers a new one. The common wisdom in general purpose expert system is pattern matching which is a necessarily lengthy procedure even for Lisp machines. In our specific application this task can be performed simply and efficiently as follows.

The basic problem boils down to the settling the two subsequent questions : given a triad, [1] in which frames does it turn up, and [2] does it couse in those frames a violation or a triggering. It is relatively little work to answer question #2 if we have a candidate list from question #1, which is the time consuming part. This is the point that our set representation comes handy. It is easy to see that a frame involves triad

[j, t, r]

if and only if all the folowing three conditions are satisfied : j belongs to its JOB_SET, t belongs to its TIME_SET and r belongs to its RESOURCE_SET.

However, checking this is easy with the above bit string representation : all we have to do is to check bit # j in the string of JOB_SET, bit # t in the string of TIME_SET and bit # t in the string of RESOURCE_SET. The answer to question [1] is affirmative only if all these three bits are 1. In Pascal, such a check is language primitive, but it can be performed by any language permitting AND operation between two strings. In the case under discussion, one string is the _SET and the other is a sting of zeroes except at position # j or t or r. If at least one AND operation results all 0, the answer to [1] is negative.

The last observation can save even more work: the lack of relevance of a frame can be established by an AND operation resulting 0 with any one _SET. Therefore it is worthwhile to perform this operation in a certain order depending on the knowledge base. We propose here such an ordering.

Consider the 0-1 string of each JOB_SET as J elements of a row in a matrix where the rows come from either the enumerative frames or the implicator part of the implicative ones. Let call the fraction of ones in the matrix as J-density, while T-density and R-density is similarly defined. Supposing that the units are randomly distributed (which is of course a rough approximation only), the the probability of obtaining a 0 by the AND operation is given by the density. Since the ANDing with a single bit at position n is equivalent with finding whether 0 or 1 is in this position, which is a random-access operation, the property of the sets for ANDing operation is irrelevant. Therefore the best is to check the strings in order of increasing density.

7. CONCLUSIONS

We described in this paper an approach for storing the knowledge base of a construction scheduling in the main memory of a personal computer which is compact and efficient to scan. To continue this research the further steps should be :

- to demonstrate on a few real-life scheduling problems that the claim that the two kinds of frames are sufficient,

- to write a code incorporating the frame compression and scanning techniques described above and

- to apply the code in actual scheduling problems evaluating its usefulness and efficincy.

We plan to continue the research in these directions in the future. Any contribution, suggestion or collaboration advancing this plan will be thankfully acknowledged.

REFERENCES

[1] Avron, B. and Feigenbaum, E. A. (eds),Handbook of Artificial Intelligence, William Kaufman, Los Altos, 1982.

[2] Bobrow, D. G. and Winograd, T., An Overview of KRL, a Knowledge Representation Language, Cognitive Science, Vol. 1, pp. 3-46 (1977).

[3] Boreland International, Turbo Pascal Version 3.0, 1985

[4] Karaa, F. A. and A. Y. Nasr, Resource Management in Construction, Journal of Construction Engineering and Management, Vol 112, pp 346-357 (1986)

[5] McGartland, M. R. and C. T. Hendrickson, Expert Systems for Construction Project Monitoring, Journal of Construction Engineering and Management, Vol 111, pp 293-307 (1985)

[6] Minsky, M.,A Framework for Representing Knowledge, in P. Winston (ed.) : The Psychology of Computers with Vision, McGraw Hill, N.Y., 1975.