Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 9 (2012) 1553 – 1562

Procedia Computer Science

International Conference on Computational Science, ICCS 2012

Integer programming applied to rule based systems

Juan Félix Ávila Herrera[1,*]

Escuela de Informática, Universidad Nacional de Costa Rica–CONICIT-MICIT

Abstract

In this paper we show how to represent a set of logic propositions as an integer linear program and how to use its solution to determine the truth value of all the proposition given only a subset of their truth values. Thus a rule–based expert system might use this approach as inference engine. When we model an expert system knowledge base as an integer linear program, we can ask for what is a minimum set of premises needed in order to have an specific conclusion as true. Furthermore we can ask for what propositions become true once a subset of propositions are known to be true. Once the integer restriction is dismissed, we can ask for what conditions are necessary in order that a conclusion holds with certain probability. Thus we can have both a deterministic and stochastic model using practically the same paradigm.

Keywords: Linear Programming, Expert Systems, Inference Engine, Logical Analysis of Data, Mathematica

1. Introduction

In artificial intelligence and specifically in expert systems theory it is very important to represent information and make deductions from this information, see [12, 15]. There are several methods to represent knowledge and using rule based systems is one of the most popular. Expert system are intended to preserve the knowledge from a human expert. Normally the most narrow or specific is the field the more easy is to represent that knowledge.

In order to develop an expert system it is necessary to address an area of interest. Once we have chosen this area of interest we need to select experts who are normally specialists capable of solving the problems in that area. For example if the area of interest is loan business a possible specialist might be an financial Analyst. After selecting the experts and they agree to give their expertise to be set in a knowledge base, a phase of "knowledge engineer" starts and its principal goal is to extract the expert expertise and represent it in a convenient format, for example (productions) rules, semantic networks, frames, scripts, etc., see [15].

As it was pointed out previously, one of the most popular methods of representation is production rules, also called rules of inference and that is the reason we call those resulting knowledge base as rule-based systems. A production rule is an expression of the form "IF (condition) THEN (conclusion)". Normally both, the condition and the conclusion in a rule are composed propositions that involve the connectives AND, OR and NOT. A production

* *Email address:* delagarita@hotmail.com (Juan Félix Ávila Herrera)
[1] Corresponding author

doi:10.1016/j.procs.2012.04.170

rule from our knowledge base becomes important in a reasoning when its condition turns out to be true. Those true conditions are called facts and they allow us to "trigger" a rule. As a consequence, the conclusion of the rule becomes part of the facts of our knowledge base and thus other rules can be triggered. We realize then that it is necessary an entity that takes a given set of facts along with our set of rules and keeps track of all the rules that are triggered and the new facts that are obtained from that process. This entity is called inference engine and can be used to decide if certain decision has to be taken or it in necessary more information to support it.

Expert systems are useful because they can help us to take a decision based on a certain number of premises. The goodness of the expert system depends on the amount of knowledge represented and the inference engine. A inference engine is a systems, normally a computer software or algorithm which let us to get deductions or "new information" from the knowledge data base. To do this in a rule–based system, we can proceed in two ways: forward chaining or backward chaining. In forward chaining each rule is examined and if it is known that its condition is true, the rule triggers and thus its conclusion is added as known data, and then process repeats iteratively.

In backward chaining we are looking for the truth value of a specific proposition. Thus we search for all the rules whose conclusions coincide this proposition. Thus we apply this idea to those conditions which implies our actual proposition and then process repeats iteratively.

When we are dealing with a rule–based system the choice of the searching method (forward or backward chaining) depends on the knowledge base. Normally what we use is backward chaining approach because we want to know if a specific proposition is true or not in order to take a decision. Eventually we can ask for additional information in order to determine if our specific proposition is true or not.

By using integer programming, in section 2 we model a set of logical propositions as an integer program and then make some deductions and observation about this approach. This set of logical propositions can be seen as a simple expert system. In section 3 we present a more interesting example that allow us to take decisions based in different conditions. In section 4 we talk briefly about Logical Analysis of Data, how we might use this technique to do automatic rule extraction and how we could use our integer programming approach to check the resulting knowledge base consistency.

Even in this paper we take advantage of some specific features of the implementation of our model, it is important to clarify that combine expert system and integer programming has been used for other researchers before. For example, Azadeh in [5] et al introduce a framework for design and optimization of strategic planning by integration of human-centered expert system and Integer Programming. Angel et all in [3] developed a hybrid tool using expert system technology and mixed-integer linear-programming models that increased fertilizer sales. McBride and O'Leary in [13] review some of that publications that put together Artificial Intelligence, Expert Systems and Mathematical programming. Dhar and Ranganathan in [8] contrast expert system and integer programming formulations of an NP-complete constraint satisfaction problem. Lee and Yong in [10] shows how to unify a linear programming model with a rule-based system by using overlapped decision variables.

2. Modeling a rule base expert system as an integer LP

We have seen that an expert system can be modeled as a set of production rules along with a set of facts. In this section we consider production rules that can be expressed in term of logical propositions. In this section we show with an example how we can use integer programming to determine what can be deduced out of this set of rules and facts. We can use a graph to create our integer program where a node represents a preposition. We will see that this graph resembles a network flow problem, however sometimes the resulting network doesn't necessarily satisfy network flow problem conditions like the one which establish that the total of incoming flow is the same that the flow that leaves the node for those nodes that are neither initial (sources) nor terminal (sinks). Remember that in a network flow problem the main goal is to try to send as much flow as possible from the source to the different destinations or sinks, see [6].

Now we are going to represent a specific rule base expert system using an integer LP. Let's suppose we have 25 propositions, namely $\{p_i : i = 1 \cdots 25\}$. Consider the next set of rules using some of this proposition:

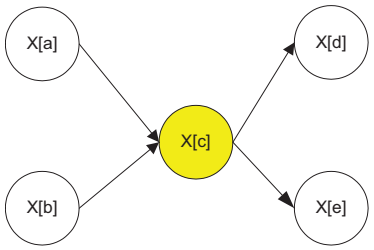

Figure 1: Representing AND and OR connectives

1. $p_1 \rightarrow p_7$
2. $p_2 \rightarrow p_8$
3. $p_3 \wedge p_4 \rightarrow p_9$
4. $p_3 \vee p_4 \rightarrow p_{10}$
5. $p_5 \rightarrow p_{11}$
6. $p_8 \wedge p_9 \rightarrow p_{13}$
7. $p_8 \vee p_9 \rightarrow p_{14}$
8. $p_{10} \wedge p_{11} \rightarrow p_{16}$
9. $p_{10} \vee p_{11} \rightarrow p_{17}$
10. $p_{13} \wedge p_{16} \rightarrow p_{19}$
11. $p_{13} \vee p_{16} \rightarrow p_{20}$
12. $p_7 \wedge p_{19} \rightarrow p_{22}$
13. $p_7 \vee p_{19} \rightarrow p_{23}$
14. $\neg p_{17} \rightarrow p_{25}$

In this example we called $\{p_i : i = 1 \cdots 5\}$ the initial propositions because they don't appear in the conclusion part of a rule. We can use a weighted digraph or network like the one[2] is shown in Figure 2 to represent this rules. We have designated with $x[i]$ a node which corresponds to the proposition p_i. Denote with $L[i, j]$ the weight of the edge $x[i] \rightarrow x[j]$. If $j \notin \{0, 6, 12, 15, 18, 21\}$ the length of the edge $x[i] \rightarrow x[j]$ represents the truth value of p_j. To do this, we assume $0 \leq L[i, j] \leq 1$. The value of $L[i, j]$ 1 is when p_j is true and 0 if not. When $0 < L[i, j] < 1$ we can suppose we are dealing with a proposition that can be partially true with probability $L[i, j]$.

Note that for nodes $x[j]$ with index $j \in \{6, 12, 15, 18, 21\}$ we have two edges coming in and two coming out. In our graph, we wil call this type of node auxiliar node. Consider a (generic) situation like the one we show in figure 1. If we require, as is suggested in [16], that $L[c, d] \leq L[c, e]$, we have that $L[c, d]$ let us compute the truth value of $p_a \wedge p_b$ and similarly $L[c, e]$, the truth value of $p_a \vee p_b$.

In order to create an integer LP which represents this set of rules we use the following decision variables for

$$i \in \{1 \cdots 24\} \setminus \{6, 12, 15, 18, 21, 24\}, \quad x[i] : \text{is 1 if } p_i \text{ is True and 0 otherwise}$$

For $i \in \{6, 12, 15, 18, 21, 24\}$ we create the auxiliary decision variables $0 \leq x[i] \leq 2$. Those auxiliary variables correspond to the auxiliary nodes we define previously. Even we are thinking in an integer LP we are going to see later on that it is possible to relax this condition, and ask for the chance or probability that some proposition was true.

Now we are going to create, using Mathematica code, an LP corresponding to our digraph. The code is presented in Fig. 3. Even it could be more didactic to present our implementation by using pseudo-code, we have used Mathematica code because it is fairly clear and those readers with access to this tool have the opportunity to reproduce our example easily. However we have written some comments after this code.

The first part of the program set the auxiliary decision variables such that its value is between 0 and 2. For the remaining variables we have that its value is between 0 and 1. The conjunction of all these conditions is called $aux1$.

Let's see what is $aux2$. When we begin setting $x[7] == x[1]$ (that comes from $p_1 \rightarrow p_7$), we note immediately that there is certain redundancy which is derived from the digraph where, by the way, we have announced the same situation. The reader can check similar redundancies in other variables but we are going to ignore this deficiency from now on. Nevertheless we have to say that in order to have a more robust model, we should set $x[7] \geq x[1]$, when we are doing forward chaining and $x[7] \leq x[1]$, when we are doing backward chaining. For simplicity, we are going to keep $x[7] == x[1]$ and do the same for analogous cases. In the example give in Section 3 we do consider this situation.

Consider now the case

$$(x[6] == x[3] + x[4]) \wedge (x[9] \leq x[10]) \wedge (x[9] + x[10] == x[3] + x[4])$$

[2]Probably the reader would suggest that it is possible to "simplify" this digraph, however we haven't done so due to didactic purposes.

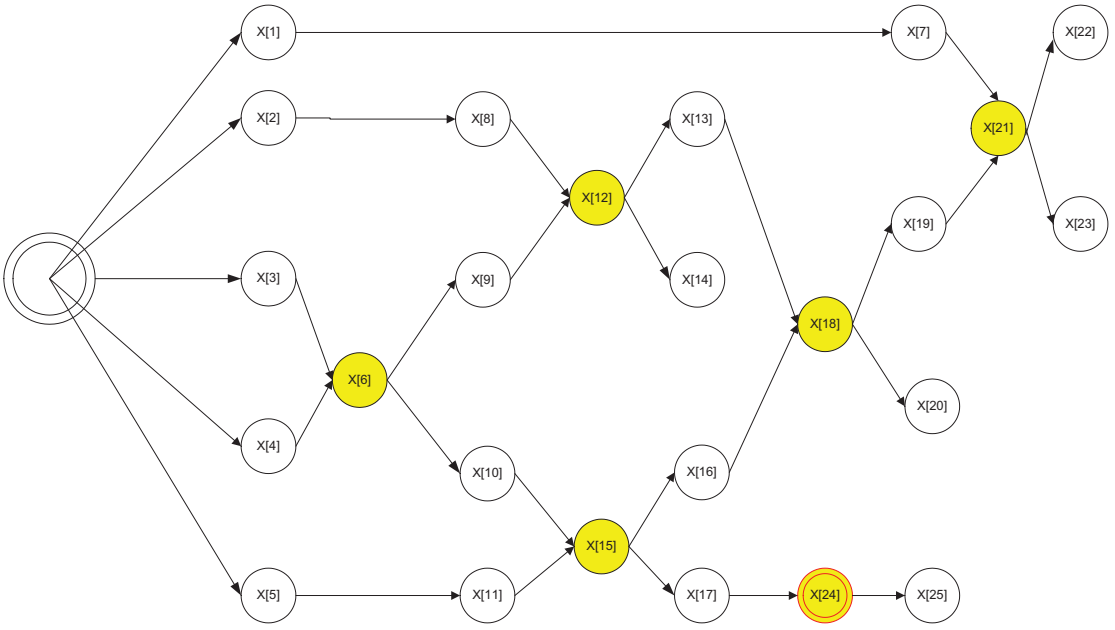

Figure 2: Digraph for the expert system

aux = True; nn = 24; $(*a*)$

For$[i = 1, i \leq \text{nn},\ \text{If}[\text{MemberQ}[\{6, 12, 15, 18, 21\}, i],\ \text{aux} = \text{aux} \wedge 0 \leq x[i] \leq 2,\ \text{aux} = \text{aux} \wedge 0 \leq x[i] \leq 1];\ i{+}{+}]$ $(*b*)$

aux1 = aux;

$$\begin{aligned}\text{aux2} = {} & (x[7] == x[1]) \wedge (x[8] == x[2]) \wedge (x[6] == x[3] + x[4]) \wedge (x[9] \leq x[10]) \wedge \\ & (x[9] + x[10] == x[3] + x[4]) \wedge (x[11] == x[5]) \wedge (x[12] == x[8] + x[9]) \wedge \\ & (x[13] \leq x[14]) \wedge (x[13] + x[14] == x[8] + x[9]) \wedge \\ & (x[15] == x[10] + x[11]) \wedge (x[10] + x[11] == x[16] + x[17]) \wedge (x[16] \leq x[17]) \ \wedge \\ & (x[18] == x[13] + x[16]) \wedge (x[19] + x[20] == x[13] + x[16]) \wedge (x[19] \leq x[20]) \wedge \\ & (x[21] == x[7] + x[19]) \wedge (x[7] + x[19] == x[22] + x[23]) \wedge (x[22] \leq x[23]) \wedge (x[24] == 1 - x[17]) \wedge \\ & (x[22] == 1)\,; \quad (*c*)\end{aligned}$$

$$\text{sol} = \text{Minimize}\left[\sum_{i=1}^{5} x[i],\ \text{aux1} \wedge \text{aux2},\ \text{Table}[x[i], \{i, 1, \text{nn}\}], \text{Integers}\right] \quad (*d*)$$

For$[i = 1, i \leq \text{Length}[\text{sol}[[2]]],$

$\text{If}[\text{sol}[[2]][[i]][[2]] \neq 0, \text{Print}[\text{sol}[[2]][[i]]]]; i{+}{+}]$ $(*e*)$

[a] Number of variables

[b] This corresponds to conditions $aux1:\ 0 \leq x_6, x_{12}, x_{15}, x_{18}, x_{21} \leq 2$ and $0 \leq x_i \leq 1$ otherwise

[c] Other conditions: $\text{aux2} = (x_7 = x_1) \wedge (x_8 = x_2) \wedge (x_6 = x_3 + x_4) \wedge (x_9 \leq x_{10}) \wedge (x_9 + x_{10} = x_3 + x_4) \wedge (x_{11} = x_5) \wedge (x_{12} = x_8 + x_9) \wedge (x_{13} \leq x_{14}) \wedge (x_{13} + x_{14} = x_8 + x_9) \wedge (x_{15} = x_{10} + x_{11}) \wedge (x_{10} + x_{11} = x_{16} + x_{17}) \wedge (x_{16} \leq x_{17}) \wedge (x_{18} = x_{13} + x_{16}) \wedge (x_{19} + x_{20} = x_{13} + x_{16}) \wedge (x_{19} \leq x_{20}) \wedge (x_{21} = x_7 + x_{19}) \wedge (x_7 + x_{19} = x_{22} + x_{23}) \wedge (x_{22} \leq x_{23}) \wedge (x_{24} = 1 - x_{17}) \wedge (x_{22} = 1);$

[d] We solve the corresponding ILP subject to the conditions $aux1$ and $aux2$

[e] We print no null variables

Figure 3: Mathematica code to solve our integer program.

This is done to correspond with the situation we have with the auxiliary node labeled with $x[6]$. Using terminology of network flows (see [6]), the idea is that the flow that arrives to the node $x[6]$ is the same that leaves the node and this flow is distributed among nodes $x[9]$ and $x[10]$. The condition $x[9] \leq x[10]$ forces $x[9]$ to have the truth value of $x[3] \wedge x[4]$ and for the same reason $x[10]$ has the truth value of $x[3] \vee x[4]$. Similar situations are repeated with $x[13]$, $x[14]$, $x[16]$, $x[17]$, $x[19]$, $x[20]$, $x[22]$ and $x[23]$.

When we set $x[22] == 1$ is because we want to know under what conditions we have that the proposition p_{22} equal to true. We note that the initial propositions that trigger the rules are $\{p_i : i = 1 \cdots 5\}$ thus it is interesting to know what is the minimum number to those propositions needed to accomplish $x[22] == 1$. In other words, we are asking what's the minimum information we have to know in order to be sure that p_{22} is true. The reader notes thus we are doing backward chaining using linear programming.

If we solve this LP using Mathematica 6.0 we get the following output.

$$\{5, \{x[1] \to 1, x[2] \to 1, x[3] \to 1, x[4] \to 1, x[5] \to 1, x[6] \to 2, x[7] \to 1, x[8] \to 1,\\ x[9] \to 1, x[10] \to 1, x[11] \to 1, x[12] \to 2, x[13] \to 1, x[14] \to 1, x[15] \to 2, x[16] \to 1,\\ x[17] \to 1, x[18] \to 2, x[19] \to 1, x[20] \to 1, x[21] \to 2, x[22] \to 1, x[23] \to 1, x[24] \to 0\}\}$$

Mathematica output presents first the optimal value of the objective function and then the corresponding assignments to each variable.

We see in this Mathematica output that it is necessary that all the initial five propositions hold true in order to have true as the truth value of p_{22}.

If we replace the condition $x[22] == 1$ with the condition $x[20] == 1$, we get

$$\{2, \{x[1] \to 0, x[2] \to 0, x[3] \to 1, x[4] \to 0, x[5] \to 1, x[6] \to 1, x[7] \to 0, x[8] \to 0,\\ x[9] \to 0, x[10] \to 1, x[11] \to 1, x[12] \to 0, x[13] \to 0, x[14] \to 0, x[15] \to 2, x[16] \to 1,\\ x[17] \to 1, x[18] \to 1, x[19] \to 0, x[20] \to 1, x[21] \to 0, x[22] \to 0, x[23] \to 0, x[24] \to 0\}\}$$

Thus only two initial propositions are necessary to be true in order to have p_{22} true.

Even more we can ask what is the minimum information necessary in order to have p_{19} true with a 85% of confidence. Now we remove the condition "Integers" from the instruction Minimize above. If we do so, we get the next solution using Mathematica 6.0:

$$\{3.4, \{x[1] \to 0., x[2] \to 1., x[3] \to 1., x[4] \to 0.4, x[5] \to 1., x[6] \to 1.4, x[7] \to 0., x[8] \to 1.,\\ x[9] \to 0.4, x[10] \to 1., x[11] \to 1., x[12] \to 1.4, x[13] \to 0.7, x[14] \to 0.7, x[15] \to 2.,\\ x[16] \to 1., x[17] \to 1., x[18] \to 1.7, x[19] \to 0.85, x[20] \to 0.85, x[21] \to 0.85,\\ x[22] \to 0.425, x[23] \to 0.425, x[24] \to 0.\}\}$$

In this solution we see that if p_2, p_3 and p_5 are true and we know that p_4 is true with probability of 40%, is reasonably to believe that p_{19} is true with probability of 85%.

Let's consider now how to do forward chaining using linear programming. Our aim is to determine what can be inferred when certain propositions are set to be true. We are interested in knowing what is the maximum number of propositions become true once some subset of them are set to be true. In Fig. 4 we present the Mathematica code to do this computation.

Our goal in this case is try to "fill" as many variables as we can with its maximum value. To accomplish this we set to 0 those variables which correspond to propositions we know are false and maximize the summation of all the decision variables. Using Mathematica 6.0 we get:

$$\{15, \{x[1] \to 0, x[2] \to 1, x[3] \to 0, x[4] \to 1, x[5] \to 1, x[6] \to 1, x[7] \to 0, x[8] \to 1,\\ x[9] \to 0, x[10] \to 1, x[11] \to 1, x[12] \to 1, x[13] \to 0, x[14] \to 1, x[15] \to 2, x[16] \to 1,\\ x[17] \to 1, x[18] \to 1, x[19] \to 0, x[20] \to 1, x[21] \to 0, x[22] \to 0, x[23] \to 0, x[24] \to 0\}\}$$

Thus if we suppose as we did in the Mathematica code that p_1, p_4 and p_5 are true, then we have, for example, that p_{20} becomes true. The maximum number of propositions that are true are 15 out of 24. If we set all the propositions p_1, $\cdots$, p_5 to be false and we ask Mathematica to solve the new integer LP we get that only p_{24} becomes true.

```
aux = True; nn = 24;
For[i = 1, i ≤ nn, If[MemberQ[{6, 12, 15, 18, 21}, i], aux = aux ∧ 0 ≤ x[i] ≤ 2,
aux = aux ∧ 0 ≤ x[i] ≤ 1]; i++]
aux1 = aux;
aux2 = (x[7] == x[1]) ∧ (x[8] == x[2]) ∧ (x[6] == x[3] + x[4]) ∧ (x[9] ≤ x[10]) ∧ (x[9] + x[10] == x[3] + x[4])
        ∧ (x[11] == x[5]) ∧ (x[12] == x[8] + x[9]) ∧ (x[13] ≤ x[14]) ∧ (x[13] + x[14]==x[8] + x[9]) ∧
       (x[15] == x[10] + x[11]) ∧ (x[10] + x[11]==x[16] + x[17]) ∧ (x[16] ≤ x[17]) ∧
       (x[18] == x[13] + x[16]) ∧ (x[19] + x[20]==x[13] + x[16]) ∧ (x[19] ≤ x[20]) ∧
       (x[21] == x[7] + x[19]) ∧ (x[7] + x[19]==x[22] + x[23]) ∧ (x[22] ≤ x[23]) ∧
       (x[24] == 1 − x[17]) ∧ (x[1] == 0) ∧ (x[3] == 0);

sol = Maximize[∑_{i=1}^{nn} x[i], aux1 ∧ aux2, Table[x[i], {i, 1, nn}], Integers]
" A solution is"
For[i = 1, i ≤ Length[sol[[2]]],
      If[sol[[2]][[i]][[2]] ≠ 0, Print[sol[[2]][[i]]]]; i++]
```

Figure 4: Forward chaining using linear programming.

If we dismiss the restriction of integer LP and open to real solutions we can ask for what happen if we know that our first 5 propositions are true with probability of 0.5. Using Mathematica 6.0 to solve this we get:

$$\{14.5, \{x[1] \to 0.5, x[2] \to 0.5, x[3] \to 0.5, x[4] \to 0.5, x[5] \to 0.5, x[6] \to 1., x[7] \to 0.5,$$
$$x[8] \to 0.5, x[9] \to 0.5, x[10] \to 0.5, x[11] \to 0.5, x[12] \to 1., x[13] \to 0.5, x[14] \to 0.5,$$
$$x[15] \to 1., x[16] \to 0.5, x[17] \to 0.5, x[18] \to 1., x[19] \to 0.5, x[20] \to 0.5, x[21] \to 1.,$$
$$x[22] \to 0.5, x[23] \to 0.5, x[24] \to 0.5\}\}$$

Thus we can interpret this solution to support, for example, that the proposition p_{18} is "sure" to be true.

3. Another example

Let's consider another simple knowledge base which rules allow us to take decisions p_6, p_7 or p_8. This time we are going to give a more simplified implementation as integer linear program.

Our set of rules is as follows:

1. $\neg p_1 \Rightarrow p_6$.
2. $p_1 \wedge p_3 \Rightarrow p_7$.
3. $p_1 \wedge \neg p_3 \Rightarrow p_8$.
4. $p_2 \Rightarrow p_1$.
5. $\neg p_2 \Rightarrow \neg p_1$.
6. $p_4 \wedge p_5 \Rightarrow p_3$.
7. $p_4 \wedge \neg p_5 \Rightarrow \neg p_3$.
8. $\neg p_4 \Rightarrow \neg p_3$.
9. $p_7 \wedge p_8 \Rightarrow \neg 1 - p_8$.

Note that in our model the initial propositions or propositions that don't appear in the left side of our rules are p_2, p_4 and p_5. Thus depending what truth values are assigned to these variables, we arrive to different conclusions.

First we suppose that our initial propositions are all true. Let's use forward chaining to determine what can be deduced from these premises and our set of rules. However, this time when we write our program, we are going to get rid of some useless variables we used in Section 2 to implement the connectives AND and NOT. We can see the Mathematica code to solve this problem in Fig. 5. An solution to this problem is as follows:

$$\{3, \{p[1] \to 1, p[2] \to 1, p[3] \to 1, p[4] \to 1, p[5] \to 1, p[6] \to 0, p[7] \to 1, p[8] \to 0, p[9] \to 1,$$
$$p[10] \to 1, p[11] \to 0, p[12] \to 1, p[13] \to 1, p[14] \to 1, p[15] \to 0, p[16] \to 1, p[17] \to 0, p[18] \to 1\}\}$$

Thus our expert system recommends p_7. If it turns out that p_2 is false, we add the condition $p[2] == 0$ to the program give in Fig. 5. In this case get:

$$\{2, \{p[1] \to 0, p[2] \to 0, p[3] \to 1, p[4] \to 1, p[5] \to 1, p[6] \to 1, p[7] \to 1, p[8] \to 0, p[9] \to 0,$$
$$p[10] \to 1, p[11] \to 0, p[12] \to 0, p[13] \to 1, p[14] \to 1, p[15] \to 0, p[16] \to 1, p[17] \to 0, p[18] \to 1\}\}$$

Thus in this case we might we can take decision p_6 or p_7.

```
aux1 = True; nn = 18;
For[i = 1, i ≤ nn, aux1 = aux1 ∧ 0 ≤ p[i] ≤ 1; i++]
aux2 = (1 − p[1] ≤ p[6]) ∧
(p[1] + p[3] == p[9] + p[10]) ∧ (p[9] ≤ p[10]) ∧ (p[9] ≤ p[7]) ∧
(p[1] + 1 − p[3] == p[11] + p[12]) ∧ (p[11] ≤ p[12]) ∧ (p[11] ≤ p[8]) ∧
(p[2] ≤ p[1]) ∧
(1 − p[2] ≤ 1 − p[1]) ∧
(p[4] + p[5] == p[13] + p[14]) ∧ (p[13] ≤ p[14]) ∧ (p[13] ≤ p[3]) ∧
(p[4] + 1 − p[5] == p[15] + p[16]) ∧ (p[15] ≤ p[16]) ∧ (p[15] ≤ 1 − p[3]) ∧
(1 − p[4] ≤ 1 − p[3]) ∧
(p[7] + p[8] == p[17] + p[18]) ∧ (p[17] ≤ p[18]) ∧ (p[17] ≤ 1 − p[18]);
sol = Maximize[p[2] + p[4] + p[5], aux1 ∧ aux2, Table[p[i], {i, 1, nn}], Integers]
aux = {};
For[i = 1, i ≤ Length[sol[[2]]],
If[sol[[2]][[i]][[2]] ≠ 0, Print[sol[[2]][[i]]]]; i++]
```

Figure 5: Mathematica code.

If we want to do backward chaining in this problem to determine under what condition we recommend p_6, we should modify the Mathematica code given in Fig. 5 by adding the restriction $p[2] + p[4] + p[5] \geq 1$ in order to have at least one of the initial preposition with truth value as true. Besides that we set $p[6] == 1$ and finally we Minimize $\sum_{i=1}^{nn} p[i]$.

One solution to this problem is as follows:

$$\{4, \{p[1] \to 0, p[2] \to 0, p[3] \to 0, p[4] \to 0, p[5] \to 1, p[6] \to 1, p[7] \to 0, p[8] \to 0, p[9] \to 0, p[10] \to 0,$$
$$p[11] \to 0, p[12] \to 1, p[13] \to 0, p[14] \to 1, p[15] \to 0, p[16] \to 0, p[17] \to 0, p[18] \to 0\}\}$$

Thus if we have that p_2 = False, p_4 = False and p_5 = True, then in that case the system would recommend p_6.

4. Automatic rule extraction

We have talked in section 1 about expert systems which expertise comes from an expert human or a group of expert humans. Once the knowledge base is built, the expert system has to be tested for errors or inconsistencies. This is normally done for a human expert. There are, however, expert system which knowledge base has been extracted automatically from a dataset. See for example [9]. Thus, we come across with rules that could be not obvious even for a human expert but are consistent with the dataset we are dealing with. In this case we can use Integer Programming approach to check the consistency of the rules we have found.

In this section we talk briefly about Logical Analysis of Data (LAD) which is a technique to do pattern recognition and how it ca be used to do automatic rule extraction. See [1, 2, 4, 7, 14].

4.1. Logical Analysis of Data

Suppose we are given a dataset D compose of o observations (e.g. patients) and each observation has v variables or features (e.g. blood pressure, age, etc). Suppose that we know the membership of each observation (e.g. sick or healthy) as well. LAD is a supervised learning technique that search for patterns or rules in this $p \times (v + 1)$ dataset that can be used to classify a new observation and assign it a risk grade. By using those (sometimes complex) rules we can define subgroups of interest within the data.

Normally LAD is used for two class dataset and hence we deal with so called positive or negative observations (that we can group as $\mathcal{P}$ and $\mathcal{N}$), however LAD might be extended to multiple class datasets. Thus LAD is suitable for clinical, genomic or proteomic studies where it is necessary to identify which combined variables are responsible for a certain condition.

LAD is a process that can be broken in five steps, namely

(a) Discretization/Binarization: This is the process of transforming numerical variables to categorical or discrete ones. Discretization increases the number of variable considerably.

(b) Support sets (best subset of variables): As a consequence of discretization process, it is necessary to select those variables that are sufficient to distinguish between negative and positive observations. This set of variables is what is called support set.

(c) Pattern generations (logical rules): Pattern generation is a process based on the use of combinatorial enumeration techniques. This process main goals are give priority to the generation of short pattern (simplicity) and cover every observation by at least one pattern.

(d) Theories and models (group of rules): The set of the positive (respectively negative) detected patterns is called a positive (respectively negative) theory. A LAD model is the union the positive and negative theories.

(e) Prediction (accuracy, forecast): Once we have a model, we are ready to make predictions based in this model.

The quality of the models depends on the quality of its patterns. For this reason it is necessary to define some concepts to measure this quality or accuracy. A pattern P is said to cover an observation $p = (x_1, \cdots, x_n)$ if p satisfies P. i.e. $P(p) = \text{True}$.

The coverage of P is defined as the number of observations covered by P. Note that P can cover either positive or negative observations. For example if $p = (1, 0, 3, 5)$ and $q = (1, 3, 2, 1)$ and $P = (x_1 > 0) \wedge (x_2 < 1)$, then P covers p but not q.

The degree of P is the number of conditions in P. For example, a typical positive pattern or rule of degree 2 could be one of the form $(x_i > 1) \wedge (x_j < 2.5)$. Thus if this condition holds for $p = (x_1, \cdots, x_n)$, then p is classify as a positive observation.

The positive (respectively negative) homogeneity of P is the ratio of number of positive (negative) observations covered by P to number of observations covered by P.

Similarly, we define the positive prevalence of P as the ratio of number of positive observations to total number of observations.

Given a model, then number of correctly classified positive observations to the number of positive observations in the test set, is said to be the sensitivity, and then number of correctly classified negative observations to number of the set of negative observations in the test set, is said to be specificity.

Finally we define the accuracy of a model as the average of sensitivity and specificity.

4.2. Ladoscope and LFW

Ladoscope is a free software (see [11]), which provides useful programs to do LAD. Ladoscope is composed of few console applications than run on Windows/Linux. Even Ladoscope is one of the most important tools to do LAD (the other one is Datascope), console applications turn out to be difficult to use special when there are a lot of parameters (pattern degree, homogeneity, prevalence, etc.) that are required to understand and aply. This is the reason why we developed LFW which is a computer software intended to be that layer that facilitates interaction between user and Ladoscope. Besides to help the user with a friendly environment, LFW let the user to take advantage of multicore or cluster architecture (generating corresponding MPI commands). This characteristic, which is not intrinsic in Ladoscope, is fundamental because by doing so, it is possible to reduce time noticeably. Beside this, once scripts are generated by LFW, they can be run without using LFW either in Linux or Windows.

When we are given a biomedical dataset it is necessary to do dimensionality reduction because this type of dataset consists of hundreds of observations and even tens of thousands of variables. In this feature selection process it is recommendable to apply some "simple" technique like for example, a t-test to detect those variables that are highly correlated with the output. There are more sophisticated approaches to do feature reduction, however they can introduce undesirable noise to the dataset that can affect or spoil it. Even though Ladoscope is (in theory) no limited by the number of variables, the more variable we use, the more time it takes and sometimes, depending of the characteristics of our computer, it can exceed their capacity and fail to complete the analysis. Due to this reason we can add another step to LAD process that we can called pre-process.

4.3. Automatic rule extraction using LAD

When we are working with a dataset, we can combine LFW and Ladoscope to get "good parameters" that give us a good LAD model and consequently a good set of patterns. A typically set of patterns given by Ladoscope are for example:

- $0, 1.0000, 0.8889, 16.0000) \quad x8 < 0.077750$
- $(0, 1.0000, 0.8889, 16.0000) \quad x8 < 0.077750 \quad \& \quad x9 < 0.108500$
- $\vdots$

Each of this lines can be broken in two parts. The first one refers to some statistics about the pattern itself, namely pattern class, homogeneity, prevalence and hazard ratio. The second part is a logical expression which describes the pattern. By using these logycal expressions we can build a set of rules that cover our training dataset and eventually might be used to do prediction for new observations.

One possible drawback about this idea is that the rules that can be found are all of the type $p \rightarrow c$ where p is a condition among the features of dataset and c is a class, namely 0 or 1. However, if our dataset is binary, we can proceed recursively by replacing the class column with each one of the features that appear in our patterns. The stopping condition of this process would be when the quality of the patterns we are getting are under certain threshold.

Due to the easy list manipulation, we can take advantage of Mathematica and create a program to convert automatically from a set of patterns given by Ladoscope into a set of rules into a linear program. This program could have options to give us the opportunity of use backward chaining or forward chaining approach.

5. Some additional considerations

Such a program would be very helpful in doing maintenance of the knowledge base. Unlike a rule–based expert system expressed in Prolog where it is easy to add or drop a rule, in the linear programming approach we need to create new decision variables and in the worst case redo the whole LP.

Another important thing to consider is how to extend this idea to predicate calculus. The use of propositions with one or more parameters give us the possibility of enlarge the scope of rule–based expert system. We leave this for future work.

It would be very interesting to apply our LP approach to games or tournaments which can be characterized in terms or rules and try to guess under what conditions a team, a sportsman or a sportswoman become a winner.

Another issue our LP approach to rule–based expert system could be valuable is checking consistency of the set of rules. Sometimes even experts can submit rules that can be contradictory. The knowledge base needs to be purged and sometimes this can take long time, expertise and patience. However using an LP (in the forward chaining) one of such inconsistencies can take us to have empty solution. However this is only an alert because normally the software that solves LP doesn't tell us what can be the cause of the inconsistency.

Due that in general solving integer linear programs could be computational costly (see [17]), it is important to recognize that our approach has this important limitation. The more rules are in knowledge base, the more difficult will be the corresponding integer program.

6. Conclusions

Rule–based expert systems are very important when we are required to preserve knowledge from an expert. When we transform one of this system in an LP we are able to answer what is the most that can be inferred from a set of propositions that are true and rules which trigger with those propositions. Similarly we can determine under what conditions we can have a proposition becomes true. This approach can be used either in a deterministic or in stochastic situation. Eventually we can use our approach to check the consistency of rules generated by an automatic rule extraction system as the one we suggest by using LAD.

References

[1] Alexe G., Alexe S., Bonates T. O., Kogan A., *Logical analysis of data - the vision of Peter L. Hammer*, Ann Math Artif Intell, USA, 2007.

[2] Alexe, G., Alexe, S., Hammer, P.L., and Vizvari, B., *Pattern-based feature selection in genomics and proteomics*, Annals of Operations Research 148, USA, 2006.

[3] Ana María Angel, Luis Alberto Taladriz and Richard Weber, *Soquimich Uses a System Based on Mixed-Integer Linear Programming and Expert Systems to Improve Customer Service*, Interfaces Vol. 33, No. 4, Chile, 2003.

[4] Reddy A. R., *Combinatorial pattern-based survival analysis with applications in Biology and Medicine*, Dissertation submitted to the Graduate School-New Brunswick Rutgers, The State University of New Jersey, USA, 2009.

[5] Azadeh, M.A.; Sharifi, S.; Izadbakhsh, H., *Integration of Expert System and Integer Programming for Optimization of Strategic Planning*, Industrial Informatics, IEEE International Conference, , 2006.

[6] Bazaraa, M.S., Jarvis, J.J., and Sherali, H., *Linear Programming and Network Flows, 2nd Edition*, John Wiley & Sons, N.Y., 1990.

[7] Boros, E.; Hammer, P.L.; Ibaraki, T.; Kogan, A.; Mayoraz, E.; Muchnik, I.;, *An implementation of logical analysis of data*, Knowledge and Data Engineering, IEEE Transactions on , 2000, 12, 2.

[8] Vasant Dhar and Nicky Ranganathan, *Integer programming vs. expert systems: an experimental comparison*, Communications of the ACM Volume 33 Issue 3, USA, 1990.

[9] Huber, K.-P.; Berthold, M.R.; , *Building precise classifiers with automatic rule extraction*, Neural Networks, 1995. Proceedings., IEEE International Conference, Pags. 1263 - 1268 vol.3, Australia, 2002.

[10] Jae Kyu Lee and Yong Uk Song, *Unification of Linear Programming with a Rule-Based System by the Post-Model Analysis Approach*, Management Science Vol. 41, No. 5, USA, 1995.

[11] Lemaire P. "Ladoscope, in Implementation of Logical Analysis of Data method" `pit.kamick.free.fr/lemaire/software-lad.html`

[12] Luger, George F., *Artificial intelligence and the design of expert systems*, Benjamin/Cummings Pub. Co., USA, 1989.

[13] Richard D. McBride, Daniel E. OLeary, *The use of mathematical programming with artificial intelligence and expert systems*, European Journal of Operational Research Volume 70, Issue 1, October, 1993.

[14] Bonates T. O. , *Optimization in logical analysis of data*, A dissertation submitted to the Graduate School-New Brunswick Rutgers, The State University of New Jersey, USA.

[15] Turban, Efraim, *Expert systems and applied artificial intelligence*, Macmillan Pub. Co., USA, 1992.

[16] V. Sgurev, *A Network Flow Approach to Clause Inference*, Problems of Engineering Cybernetics and Robotics, 46, Bulgarian Academy of Sciences, 1997.

[17] Alexander Schrijver, *Theory of Linear and Integer Programming*, Wiley, USA, 1998.