БЪЛГАРСКА АКАДЕМИЯ НА НАУКИТЕ . BULGARIAN ACADEMY OF SCIENCES

ПРОБЛЕМИ НА ТЕХНИЧЕСКАТА КИБЕРНЕТИКА И РОБОТИКАТА, 46
PROBLEMS OF ENGINEERING CYBERNETICS AND ROBOTICS, 46

София . 1997 . Sofia

A Network Flow Approach to Clause Inference*

Vassil Sgurev

Institute of Information Technologies, 1113 Sofia

1. Introduction

The clause inference based on the resolution principles of J. Robinson [1] plays a fundamental role in different logical programming systems and in artificial intelligence as a whole. This role is illustrated completely and successfully enough in the well-known book of R. Kowalski [2].

During the second half of the 80-ies, various quantitative methods have been suggested interpreting the logical, and particularly the clause inference, reducing this inference to solving problems of integer and/or binary programming [3].

Not setting forward all the positive aspects of the quantitative approaches suggested, we have to note, that they possess a number of inconveniences and shortcomings, connected with the lack of an efficient graphic illustration of the inference and the impossibility for satisfactory representation of the predicate properties with their help.

The present paper suggests a new quantitative approach towards clause inference, based on the application of a special class of network flows, which in our opinion avoids the limitations above mentioned and realizes efficient quantitative representation of the clause inference in the form of a flow on a graph.

Unlike the graphic interpretations presented up to now, the logic operations, the resolution and the inference are realized through the network nodes, and the atomic formulae and clauses are regarded as directed arcs in the same network. The flow functions on these arcs accept values of either zero (false) or one (truth).

2. Network flows

A Directed Network, or a Directed Graph $G=[N,U]$ consists of a set N of elements $x, y, \ldots,$ and of a set U of ordered pairs (x,y) of elements of N. The elements of the set N are called vertices or nodes, those of U – arcs or edges.

* The present paper is written in conformance with contract No И-605 with the National Fund for Scientific Investigations at the Bulgarian Ministry of Education and Technologies.

If $x \in N$, then $A(x)$ (i.e. After x) is the set $y \in N$ for which $(x,y) \in U$:

$$A(x) = \{y \in N \,/\, (x,y) \in U\},$$

$$B(x) = \{y \in N \,/\, (y,x) \in U\} \quad \text{(Before } x\text{)}.$$

Let s and t be two nodes from N. The stationary flow of a value of v, from s towards t in the network $[N,U]$ is a function f over U with values into the set of Non-negative integers, satisfying the linear equalities and inequalities:

(2.1) $$\sum_{y \in A(x)} f(x,y) - \sum_{y \in B(x)} f(y,x) = \begin{cases} v, & x = s, \\ 0, & x \neq s, t, \\ -v, & x = t. \end{cases}$$

(2.2) $$f(x,y) \leq c(x,y) \text{ for all } (x,y) \in U,$$

(2.3) $$f(x,y) \geq 0 \text{ for all } (x,y) \in U.$$

The variables $f(x,y)$ have integer values.

We call s – a source, and t – a sink, (2.1) is called equation of conservation, the function $c(x,y)$ is the arc capacity of the arc (x,y). If the flow f is given, the number $f(x,y)$ is called the arc flow on arc (x,y).

The network flow (2.1)–(2.3) is known as classical flow and was introduced by Ford and Fulkerson [4].

For the representation of the clause inference we need a more complicated class of network flows called in [5] linear flows. In these the relatively simple way of giving the arc capacity constraint of the Ford and Fulkerson flows by equation (2.2) is replaced by a definition using linear constraints:

(2.4) $$\sum_{(x,y) \in D_i} b_i(x,y)\, f(x,y) \leq C_i; \quad i \in I = \{1, 2, \ldots, k\},$$

where

$$b_i(x,y) \begin{cases} \in R', \text{ if } i \in I \text{ and } (x,y) \in D_i, \\ = 0 \text{ otherwise;} \end{cases}$$

R' is the set of the real numbers different from 0;

$C_i \in R'$, $C_i > 0$;

D_i are non-intersecting subsets of arcs, for which for any

(2.5) $$i, j \in I, \quad D_i \cap D_j = \emptyset; \quad \bigcup_{i \in I} (D_i) = U.$$

The basic properties of the linear flow (2.1), (2.3), (2.4) and approximative and precise methods for solving the optimization problems associated to these flows are given in [5].

3. Network flow representation of clauses

It is possible to find out network flow representations for each operator (and, or, not, implies) of propositional logic. In this paper we have focused our attention on clausal form propositions because of their large application in logic programming.

3.1. Clauses

A clause, in propositional logic, could be defined as an expression of the form

$$B_1, B_2, \ldots, B_m \leftarrow A_1, A_2, \ldots, A_n,$$

where A_i and B_i are atomic formulae. $A_1, \ldots, A_n$ are the joint conditions of the clause and $B_1, B_2, \ldots, B_m$ are the alternative conclusions.

If $n=0$, it states that: B_1 or B_2 or ... or B_m.

If $m=0$, it states that, it is **not** the case that:

$$A_1 \text{ and } A_2 \text{ and } \ldots \text{ and } A_n.$$

If $m=n=0$ then it is an empty clause and is interpreted as a sentence being always false.

An atomic formula is an expression of the form P where P is a propositional symbol.

$B \leftarrow A$ (B if A) corresponds to the more familiar $A \rightarrow B$ (if A then B) and is used in order to draw attention to the conclusion.

3.2. Network flow representation of a clause

A clause $B \leftarrow A$, or the equivalent $A \rightarrow B$, could be represented in terms of network flows as a block consisting in a node, the associated arcs and some constraints which guarantee the obtaining of arc flows corresponding to the values of the truth table of the implication. This table is

A	B	$(A \rightarrow B)$
F	F	T
F	T	T
T	T	T
T	F	F

The logical constant True (T) is interpreted as an arc flow having the value 1, F – as a flow equal to 0. Of course several different representations are possible. We have chosen to consider a case corresponding to the Modus Ponens inference rule:

$$A \rightarrow B, A, \text{ infers } B.$$

We will have the values of $(A \rightarrow B)$ and A at the input of the node, the value of B being at the output. The resulting piece of network is as in Fig. 1.

Fig. 1

$f(a,b)$ corresponding to the truth value of A, $f(c,b)$ to the truth value of $(A \rightarrow B)$, $f(b,y)$ to the truth value of B. According to the conservation equation the input flow in the node must be equal to the output flow and we need an arc (b, t_0) to evacuate the extra flow occurring in some cases.

Thus we have the following table of the desired values of the arc flows:

$f(c,b)$	$f(a,b)$	$f(b,y)$	$f(b,t_0)$
1	0	1	0
1	0	0	1
1	1	1	1
0	1	0	1

The third line corresponds to Modus Ponens case.

The constraints that are necessary to obtain these values are:

(3.1) $$2f(b,y) - f(c,b) \leq 1,$$

(3.2) $$f(b,t_0) \leq 1.$$

We need another constraint in order to exclude the possibility of having both $f(a,b)$ and $f(c,b)$ equal to 0 (i.e. having both A and $(A \to B)$ at false, which is impossible according to the truth table of the implication). Such constraint is:

(3.3) $$f(a,b) + f(c,b) \geq 1,$$

where $f(c,b)$, $f(a,b)$, $f(b,y)$ and $f(b,t_0)$ are integer numbers.

Our next step is to consider Horn clauses having more than one condition and only one conclusion:

$$B \leftarrow A_1, \ldots, A_n.$$

We have focused our attention on these clauses as being simpler and very useful in logic programming. The node corresponding to a Horn clause is as in Fig. 2.

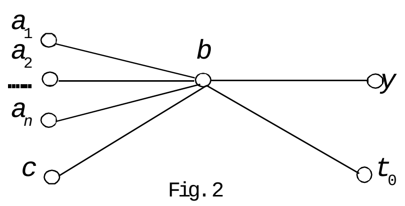

Fig. 2

Thus we have the following table of the desired values:

$f(c,b)$	$f(a_1,b)$	$f(a_n,b)$	$f(b,y)$	$f(b,t_0)$
1	0	–	1	$<n$
1	–	0	0	$\leq n$
1	1	…1	1	n
0	1	…1	0	n

Here the constraints do not vary much as the first one doesn't depend on the number of inputs. This first constraint is

(3.4) $$f(b,y) \leq \frac{f(c,b) + 1}{2}.$$

We can write it, together with the other two constraints, in a way more appropriate to mathematical programming:

(3.5) $$2\,f(b,y) - f(c,b) \leq 1,$$

(3.6) $$f(b,t_0) \leq n,$$

(n is the number of conditions),

(3.7) $$f(c,b)+\sum_{i=1}^{n}\frac{f(a_i,b)}{n}\geq 1.$$

In order to represent a general Non-Horn clause with more than one conclusion an additional node S_0, which supplies some flow if the available input flow is not sufficient, is needed. In the corresponding network we have now $n+2$ input arcs and $m+1$ output arcs (Fig. 3).

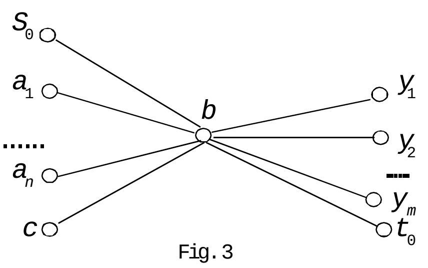

Fig. 3

The table of the desired values is:

$f(c,b)$	$f(a_1,b)$	$f(a_n,b)$	$\bigvee_{j=1}^{m}(f(b,y_i))$	$f(b,t_0)$
1	0	–	1	$<n$
1	–	0	0	$\leq n$
1	1	…1	1	$\leq n$
0	1	…1	0	n

$\bigvee_{j=1}^{m}(f(b,y_i))$ is equal to 0 if all the $f(b,y_j)$ are equal to 0, and is equal to 1 if at least one of the $f(b,y_j)$ are equal to 1 ($B_1, \ldots, B_m$ being a disjunction of conclusions). The constraints are:

(3.8) $$2\,f(b,y_j)-f(c,b)\leq 1 \text{ for } j=1 \text{ to } m,$$

(3.9) $$f(b,t_0)\leq n,$$

(3.10) $$f(c,b)+\sum_{i=1}^{n}\frac{f(a_i,b)}{n}\geq 1.$$

In this case we are interested in minimizing the flow $f(S_0,i)$ (it isn't absolutely necessary as both $f(b,t_0)$ and $f(i,b_j)$ are bound) so as to limit the unusable flow. This might be done with an optional constraint like:

(3.11) $$f(S_0,i)\leq m-1,$$

which overvalues $f(S_0,i)$.

3.3. Some special clauses

Horn clauses of the kind: $B\leftarrow$ are interpreted as assertions or as facts and, in order to be represented in flow terms, could be viewed as $B\leftarrow$True. Knowing that

(3.12) $$f(S_0,i)=1$$

and using the same constraints as above, the network might be as in Fig. 4.

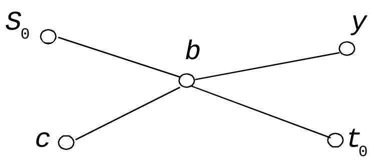

Fig. 4

The same clause could be represented just as an arc, at the beginning of the network, having a flow of 1 (Fig. 5).

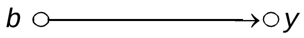

Fig. 5

(3.13) $$f(b,y) = 1$$

Clauses of the kind: $\leftarrow A_1, \ldots, A_n$ are interpreted as denials and could be viewed as

$$\text{False} \leftarrow A_1, \ldots, A_n.$$

They can be represented in flow terms like a clause node having 0 at his output (Fig.6).

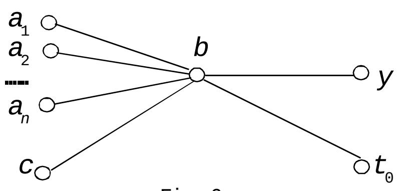

Fig. 6

The constraints being:

(3.14) $$f(b,y) = 0,$$

(3.15) $$f(b,t_0) < n.$$

We don't treat in that way the denial representing the goal to prove. Instead we treat it as a (positive) question which answer (0 or 1, i.e. True or False) is to be find.

3.4. Multiplication node

We introduce here an additional node which has no logical equivalent and which is aimed to multiply the value of an arc flow, representing a certain proposition, the number of times this proposition occurs in the considered formula, e.g. in:

$$B \leftarrow A_1, A_2$$

$$D \leftarrow A_2, C_1, C_2$$

A_2 has two occurrences, so the value of the corresponding flow must be multiplied twice.

The multiplication node will "copy" the value of the input arc flow in each output arc. The additional flow needed to satisfy the conservation equation is provided by a node S_0.

$f(S_0,b)$	$f(a,b)$	$f(b,y_1)$	$f(b,y_2)$		$f(b,y_n)$
0	0	0	0	...	0
$n-1$	1	1	1	...	0

and the network representation is as in Fig. 7.

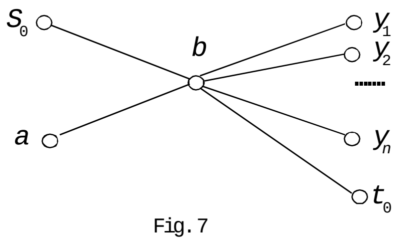

Fig. 7

The constraints are the following:

(3.16) $f(b,y_j) - f(a,b) = 0$ for $i=1$ to n
(n constraints).

4. Network Flow Representation (in general)

On the basis of the network flow representation of the separate clauses and the atomic formulae, described in chapter 3, and depending on the initial set of clauses and/or atomic formulae, this set and the clauses investigated can be represented with the help of the linear network flow (2.1), (2.3) and (2.4). In place of the linear constraints, written in their general form, the different constraints (3.1)–(3.16) are used instead.

In this approach one and the same clauses are represented by one and the same graph arc, if possible, or by different arcs, with guaranteed equality of the flow functions.

Let U' denote the set only of those arcs in the network, with the help of which clauses or atomic formulae are denoted. Then solving the optimization problem given below with the help of the specific network flow methods:

(4.1) $$\max \sum_{(x,y)\in U'} f(x,y)$$

subject to constraints (2.1), (2.3) and the corresponding relations (3.1)–(3.16), the exact states "truth-false" will be obtained for each one of the literals considered – depending on the value one or zero of the respective arc flow function.

In case the problem above described has no solution, the set of clauses is a contradictory one.

There is no doubt, that the network flow discussed can be applied in an analogous way for propositional logic interpretation also, but this aspect is not a subject of the present paper.

The study of the arc capacity of the logical flow obtained is of future interest, as well as the possibility to transfer some of the network flow results to clause inference.

The application of the network flow approach in logic programming requires the development of some efficient techniques and methods for this purpose.

References

1. Robinson, J. A. A machine oriented logic based on the resolution principle. – J.ACM, **18** (January), 1965, 23–41.
2. Kowalski, R. A. Logic for Problem Solving. Elsevier North Holland, New York, 1979.
3. Hooker, J. N. A quantitative approach to logical inference. – Decision Support Systems, Vol.4, 1988, No 1, 45–69.
4. Ford, L. R., D. R. Fulkerson. Flows in Networks. Princeton N.J., Princeton University Press, 1962.
5. Sgurev, V. Network Flows with General Constraints. Publishing House of the Bulgarian Academie of Sciences, Sofia, 1991.

Сетевой поток– подход для клаузиального вывода

Васил Сгурев

Институт информационных технологий, 1113 София

(Р е з ю м е)

Настоящая статья посвещена возможности представления логического вывода для клауз Хорна как особый класс целочисленного сетевого потока с дополни-тельными равенствами и неравенствами. Возможность такого представления позволяет рассматривать этот вывод как экстремальная задача в моделях исследований операций.

Предлагаемый класс сетевых потоков для интерпретации вывода имеет ряд особеностей, которые существено отличают его от классического потока Форда и Фалкерсона.

В работе предложены инструментальные средства, с помощью которых и на базе указанного сетевого потока из баз знаний с истинными утвреждениями и условиями "если–то" можно извлечь новые знания.