CIMCA 2001 Proceedings/ISBN 0858898470: M. Mohammadian (Ed.)

An Efficient Algorithm for Approximate Reasoning with Multiple Antecedents

P.M. van den Broek
Department of Computer Science, University of Twente,
P.O.Box 217, 7500 AE Enschede, the Netherlands
E-mail: pimvdb@cs.utwente.nl

Abstract

We present an efficient algorithm for approximate reasoning with multiple antecedents. This algorithm computes its result by aggregating results from single antecedent rules. We prove its correctness, implement it in the functional programming language Miranda, and show its effectiveness with an example.

1 Introduction

In [1,2,3] we have presented efficient algorithms for approximate reasoning, in case the rules have a single antecedent. In this paper we present an efficient algorithm which deals with multiple antecedents. Consider for example a rule with two antecedents:

Rule	IF	X=A	AND	Y=B THEN	Z = C
Facts		X=A’	AND	Y=B’	

Conclusion	Z = C’

Here A and A’ are fuzzy sets on a universe U1, B and B’ are fuzzy sets on a universe U2, and C and C’ are fuzzy sets on a universe U3.
The (membership function of the) resulting fuzzy set C’ is defined by

$$C'(z) = \sup_{x,y} \min (\min (A'(x),B'(y)), J (\min (A(x),B(y)), C(z))) \tag{1}$$

where J is some implication operator.

From a computational point of view, Eq. (1) has two important drawbacks. In the first place, the supremum is taken over the whole Cartesean product space U1xU2. This is not efficient; it leads to a complexity which grows exponentially with the number of antecedents. In the second place, if A and B are continuous and piecewise linear on intervals U1 and U2 respectively, the same is not true for their product on U1xU2; this means that the algorithms in [1], devised to be used in case of continuous piecewise linear membership functions, cannot be used in case of multiple antecedents.

Both these drawbacks of Eq. (1) would have been overcome if we would have the following equality:

$$C'(z) = \max (C_1(z),C_2(z)) \tag{2}$$

where the functions C_1 and C_2 are defined by

$$C_1(z) = \sup_x \min (A'(x), J (A(x), C(z))) \tag{3}$$

$C_2(z) = \sup_y \min (B'(y), J (B(y), C(z)))$ (4)

The fuzzy sets C_1 and C_2 are the result from the rules with only the first respectively. the second antecedent; Eq. (2) then expresses that the result C' is aggregated with the maximum operator from the results C_1 and C_2.

The problem is, however, that Eq. (2) does not hold in general. Consider for example the case where U1 consists of the single element x, U2 consists of the single element y, and U3 consists of the single element z. Then U1xU2 consists of the single element (x,y). Let A,A',B,B' and C be given by $A'(x) = 0$ and $A(x) = B(y) = B'(y) = C(z) = 1$. Since $J(1,1) = 1$ one verifies immediately that $C'(z) = C_1(z) = 0$ and $C_2(z) = 1$, contradicting Eq. (2).

The good news, on the other hand, is that under some mild conditions, Eq. (2) does hold. These conditions are that the implication operator J is monotonically non-increasing in its first argument, and the input fuzzy sets A' and B' are normal, i.e. their membership functions take the value 1 somewhere on their domain.

Section 2 contains our main results with their proofs, and in section 3 we implement them by giving, in the functional programming language Miranda [4], several higher order functions which return a function for multiple antecedent approximate reasoning, when given a function for single antecedent approximate reasoning as input.

2 Approximate reasoning with multiple antecedents

We will consider here the case of approximate reasoning with generalised modus ponens where a rule has N antecedents:

Rule	IF	$X_1 = A_1$ AND	.....	AND	$X_N = A_N$ THEN	$Y = B$
Facts		$X_1 = A_1'$ AND	.....	AND	$X_N = A_N'$	
Conclusion						$Y = B'$

The resulting fuzzy set B' is defined by

$$B'(y) = \sup_{x1..xN} \min (\min_i \{A_i'(x_i)\}, J (\min_i \{A_i(x_i)\}, B(y))) \quad (5)$$

where J is some implication operator, the index i ranges from 1 to N, and the variable x_i ranges over the universe U_i.

The main result of this paper is that Eq. (5) is equivalent to the equation

$$B'(y) = \max_i \{B_i(y)\} \quad (6)$$

where

$$B_i(y) = \sup_x \min (A_i'(x), J (A_i(x), B(y))) \quad (7)$$

if the following two conditions are satisfied:

- The implication operator J is monotonically non-increasing in its first argument.

- The input fuzzy sets A_i' are normal, i.e. their membership functions take the value 1 somewhere on their domain.

These conditions are rather mild. Indeed, most of the implication operators listed by Klir & Yuan [5] are monotonically non-increasing in their first argument, and normality is a quite common condition on fuzzy sets.
If not all input sets are normal, instead of Eq. (6), we have

$$B'(y) = \min (\max_i \{B_i(y)\}, \min_i \{\sup_x (A_i'(x)\}) \quad (8)$$

This equation states that the inference result is obtained by first computing the inference result just as for normal input sets, and then taking the minimum of the result and the smallest supremum of the input sets.

To prove these results, we start with Eq. (5). Since J is monotonically non-increasing in its first argument, it can be rewritten to

$$B'(y) = \sup_{x1..xN} \min (\min_i \{A_i'(x_i)\}, \max_i \{ J (A_i(x_i), B(y)) \}) \quad (9)$$

Since the minimum operator is monotonically non-decreasing in its second argument we can rewrite this equation to

$$B'(y) = \sup_{x1..xN} \max_j \{ \min (\min_i \{A_i'(x_i)\}, J (A_j(x_j), B(y))) \} \quad (10)$$

In the next step we use the fact that $\sup_x \max_i \{f_i(x)\} = \max_i \{\sup_x f_i(x)\}$ for each set of functions $\{f_i\}$. We obtain

$$B'(y) = \max_j \{ \sup_{x1..xN} \min (\min_i \{A_i'(x_i)\}, J (A_j(x_j), B(y))) \} \quad (11)$$

In the following step we use the fact that $\sup_x \min (f(x), g) = \min (\sup_x f(x),g)$ for all functions f and all numbers g. We obtain

$$B'(y) = \max_j \{ \sup_{xj} \min (\sup_{x1..xN-xj} \min_i \{A_i'(x_i)\}, J (A_j(x_j), B(y))) \} \quad (12)$$

Here $\sup_{x1..xN-xj}$ denotes the supremum over all variables $x_1 .. x_N$, with the exception of x_j.
The final step is to use the normality of the fuzzy sets $\{A_i'\}$ to conclude that

$$\sup_{x1..xN-xj} \min_i \{A_i'(x_i)\} = A_j'(x_j). \quad (13)$$

So we obtain

$$B'(y) = \max_j \{ \sup_{xj} \min (A_j'(x_j), J (A_j(x_j), B(y))) \} \quad (14)$$

which proves the eqs. (6) and (7).

If not all the fuzzy sets A_i' are normal, we use

$$\sup_{x1..xN-xj} \min_i \{A_i'(x_i)\} = \min (A_j'(x_j), \min_{i\neq j}\{ \sup_{xi} A_i'(x_i)\}) \quad (15)$$

to rewrite Eq.(12) to

$$B'(y) = \max_j \{\sup_{x_j} \min (\min_{i \neq j}\{ \sup_{x_i} A_i'(x_i)\}, \min (A_j'(x_j), J (A_j(x_j), B(y)))) \} \quad (16)$$

In this equation we can replace $\min_{i \neq j}\{ \sup_{x_i} A_i'(x_i)\}$ by $\min_i\{ \sup_{x_i} A_i'(x_i)\}$, since $\sup_{x_j} A_j'(x_j)$ is not less than $\min (A_j'(x_j), J (A_j(x_j), B(y)))$. Then Eq. (8) is derived immediately.

In case J is a t-norm (with the Mamdani minimum operator as well known example) instead of an implication operator, the Eqs (6) and (7) hold when in Eq. (6) max is replaced by min. In this case no conditions on the t-norm and the input fuzzy sets need to be imposed. For the special cases where J is the minimum operator (Mamdani) or the multiplication operator (Larsen), this result is known already; see for instance [6].

The proof is as follows. Since by definition a t-norm is monotonically non-decreasing in its first argument [5], from Eq. (5) we obtain Eq. (9) with max replaced by min:

$$B'(y) = \sup_{x1..xN} \min (\min_i \{A_i'(x_i)\}, \min_i \{ J (A_i(x_i), B(y)) \}) \quad (17)$$

The former proof is not applicable here, since there is no step analogous to the step from Eq. (10) to Eq. (11). However, rearranging terms in Eq. (14) gives

$$B'(y) = \sup_{x1..xN} \min_i \{ \min (A_i'(x_i), J (A_i(x_i), B(y))) \} \quad (18)$$

and this can be rewritten to

$$B'(y) = \min_i \{ \sup_{x_i} \min (A_i'(x_i), J (A_i(x_i), B(y))) \} \quad (19)$$

which proves the Eqs. (6) and (7), with max replaced by min.

3 Implementation and example

In this section we will give implementations of the algorithm in the functional programming language Miranda [4]. In [1] we have given an Miranda implementation of approximate reasoning for rules with a single antecedent for the case the universes are intervals of the real numbers, for several implication operators. In [3] we treated the case where the universes are finite. In order not to repeat these implementations in this paper, we will give, for both cases, a function which takes a single antecedent approximate reasoning function as an argument, and returns a corresponding multiple antecedent approximate reasoning function as a result.

First suppose that all universes are finite. In this case our Miranda implementation is:

```
appr_reas == [num] -> [num] -> [num] -> [num]
appr_reas_mult == [[num]] -> [[num]] -> [num] -> [num]

mult_ants :: appr_reas -> appr_reas_mult
mult_ants f as a's b
  = aggregate [f a a' b |(a,a')<-zip (as,a's)]
    where aggregate prs
              = [], if hd prs = []
              = max (map hd prs) : aggregate (map tl prs), otherwise
```

Here `appr_reas` is the type of functions for approximate reasoning with a single antecedent. The first parameter is the antecedent of the rule, which is a list of membership values. The second parameter is the input fuzzy set, the third parameter is the consequent of the rule. The type `appr_reas` is the type of functions for approximate reasoning with a multiple antecedents. The first parameter is a list of antecedents of the rule,the second parameter is a list of input fuzzy sets, the third parameter is the consequent of the rule. The function `mult_ants` maps an `appr_reas` to an `appr_reas_mult`, as described in the previous section.

As an example, consider the case where the number of antecedents is 3, and all universes have 6 elements. All fuzzy sets are denoted as a list with 6 numbers. Let the antecedent fuzzy sets a1, a2 and a3 be given by

```
a1 = [0.5,1,1,0.5,0.5,0]
a2 = [0,0.5,1,1,0.5,0]
a3 = [0,0,0.5,1,1,0.5]
```

the corresponding (normal) input fuzzy sets a1', a2' and a3' by

```
a1' = [0,0,0.5,1,1,0.5]
a2' = [0.5,1,1,0.5,0.5,0]
a3' = [0,0.5,1,1,0.5,0]
```

and the output fuzzy set b by

```
b = [0,0.2,0.4,0.6,0.8,1]
```

In [3] we derived the function `kl_d2`, for efficient approximate reasoning with a single antecedent using the Kleene Dienes implication. This function can now be used to perform the calculation of the inference result in the multi antecedent case : the expression to be evaluated is `mult_ants kl_d2 [a1,a2,a3] [a1',a2',a3'] b` , and the evaluation result is `[0.5,0.5,0.5,0.6,0.8,1]`.

To judge the efficiency of this calculation we need to compare it with a straightforward calculation. The straightforward calculation can be done with the function `mult_ants2`, which implements eq. (5) and is given by

```
mult_ants2 f as a's b = f (c as) (c a's) b
                        where c [x] = x
                              c (x:ys) = [min [p,q] | p<-x; q<-c ys]
```

The calculation with `mult_ants` used 1103 reductions, whereas the calculation with `mult_ants2` used 13198 reductions. This speed up factor of about twelve is what we expect: `mult_ants2` performs a computation of linear complexity over a Cartesian product space of 6^3 = 216 elements, whereas `mult_ants` performs three times a linear computation over a space of 6 elements.

With the Early Zadeh implication operator (the function `ez2` in [3]) the number of reductions is 2056 and 25001 respectively, and with the Willmott implication operator (the function `willmott2` in [3]) the number of reductions is 2699 and 25218 respectively.

Now suppose that all universes are intervals. In this case our Miranda implementation is:

```
appr_reas == plf -> plf -> plf -> plf
appr_reas_mult == [plf] -> [plf] -> plf -> plf

mult_ants :: appr_rea -> appr_rea_mult
mult_ants f as a's b
  = aggregate [f a a' b| (a,a') <- zip (as,a's)]
    where
    aggregate [x] = x
    aggregate (x:xs) = maxplf x (aggregate xs)
```

Here `plf` denotes the type of continuous piecewise linear membership functions, as described in [1]. This implementation is similar to the previous one. The function `maxplf` returns the maximum of two `plf`'s, and is described in [1].

As an example, consider the case where the number of antecedents is 3, and all universes are the interval [0..5]. All fuzzy sets are denoted as a list with pairs of numbers, the first number denoting a point of the interval, and the second number denoting the value of the membership function in that point. The membership function is linear between these points .

Let the antecedent fuzzy sets a1, a2 and a3 be given by

```
a1 = [(0,0.5),(1,1),(2,1),(3,0.5),(4,0.5),(5,0)]
a2 = [(0,0),(1,0.5),(2,1),(3,1),(4,0.5),(5,0)]
a3 = [(0,0),(1,0),(2,0.5),(3,1),(4,1),(5,0.5)]
```

the corresponding (normal) input fuzzy sets a1', a2' and a3' by

```
a1' = [0,0,0.5,1,1,0.5] [(0,0),(1,0),(2,0.5),(3,1),(4,1),(5,0.5)]
a2' = [0.5,1,1,0.5,0.5,0][(0,0.5),(1,1),(2,1),(3,0.5),(4,0.5),(5,0)]
a3' = [0,0.5,1,1,0.5,0][(0,0),(1,0.5),(2,1),(3,1),(4,0.5),(5,0)]
```

and the output fuzzy set b by

```
b = [(0,0),(1,0.2),(2,0.4),(3,0.6),(4,0.8),(5,1)]
```

In [1] we derived the function `kleene_dienes`, for approximate reasoning with a single antecedent using the Kleene Dienes implication. This function can now be used to perform the calculation of the inference result in the multi antecedent case : the expression to be evaluated is:

```
mult_ants (develop kleene_dienes) [a1,a2,a3] [a1',a2',a3'] b.
```

Here the function `develop` is defined by

```
develop f a a’ b = compose (f a a’) b
```

and the function `compose` is defined in [1]. The evaluation result is `[(0,0.75),(3.75,0.75),(5,1.0)]`.

The efficiency of this algorithm cannot be compared with the efficiency of a straightforward algorithm as in the case of finite universes, since in this case no straightforward algorithm exists.

4 Conclusion

We presented an efficient algorithm for approximate reasoning with multiple antecedents. This algorithm computes its result by aggregating results from single antecedent rules. We proved its correctness, implemented it in the functional programming language Miranda, and showed its effectiveness with an example.

References

[1] P.M. van den Broek, Fuzzy Reasoning with Continuous Piecewise Linear Membership Functions, *Proceedings of the 1997 Annual Meeting of the North American Fuzzy Information Processing Society (NAFIPS'97),* C. Isik & V. Cross (Eds), IEEE Catalog Number 97TH8297, 371-376, 1997.

[2] P.M. van den Broek, Multiconditional Approximate Reasoning with Continuous Piecewise Linear Membership Functions, *Computational Intelligence for Modelling, Control & Automation; Evolutionary Computation & Fuzzy Logic for Intelligent Control, Knowledge Acquisition & Information Retrieval (CIMCA'99),* M. Mohammadian (Ed), IOS Press, 246-253, 1999.

[3] P.M. van den Broek, Efficient Algorithms for Approximate Reasoning, *Proceedings of the 6th International Conference on Neural Information Processing (ICONIP'99)*, T. Gedeon, P.Wong, S. Halgamuge, N.Kasabov, D.Nauck & K.Fukushima (Eds), ISBN 0-7803-5871-6, 292-297, 1999.

[4] D. Turner, Miranda: a non-strict functional language with functional types, *Functional Programming Languages and Computer Architecture,* Lecture Notes in Computer Science Vol. 201, J.-P. Jouannaud (Ed), Springer Verlag, 1 – 16, 1985.

[5] G.J. Klir & B. Yuan, *Fuzzy sets and fuzzy logic, theory and applications,* Prentice Hall 1995.

[6] W. Pedrycz & F. Gomide, *An Introduction to Fuzzy Sets*, MIT Press, 1998.