IS-FBN, IS-FBS, IS-IAC: The Adaptation of Two Classic Algorithms for the Generation of Referring Expressions in order to Produce Expressions like Humans Do

Bernd Bohnet
Innovative Language Technology, TTI GmbH
and
University of Stuttgart
Visualization and Interactive Systems Group
Universitätstr. 58, 70569 Stuttgart, Germany
bohnet@informatik.uni-stuttgart.de

Abstract

The first algorithm is the full brevity algorithm combined with nearest neighbour learning technique which is used for the selection of the referring expression closest to training examples as uttered by humans. This algorithm obviously only imitates human behaviour with the goal to get high scores. With the Incremental Algorithm that is known and justified by its human like behaviour, we tried to beat the first one.

1 Introduction

The first Shared Task and Evaluation Campaign (STEC) in Natural Language Generation is the background of this work.

In section 2, we adopted from Bohnet and Dale (2005) the description of problem solving by search in order to describe the methods as used in the algorithms that we contribute to the shared task. In section 3, we provide the evaluation and results gained from the development data.

2 Description of the Methods

Definition 1 shows the node and state classes as well as the definition of the basic algorithm for problem solving by search. In contrast to the original definition of the algorithm by Simon and Newell (1963) in a functional programming language, we adopt here an object oriented formalism, since this allows the representation of dependency between the algorithms by means of inheritance and overwriting.

Full Brevity using Nearest Neighbour. Definition 2 shows the Full Brevity Algorithm (**FB**) that computes all combinations with increasing length. The first combination that fulfils the goal is the shortest referring expression or one of the shortest possible referring expressions.

Definition 1: The Node and State Classes and the Basic Algorithm Structure

```
class Node {
  $s$ // State
  $getState()$ {return $s$} // returns the state of the node
}
class State {
  $L$ // Set of chosen properties and/or relations
  $C$ // Set of distractors
  $P$ // Set of available properties and/or relations
}
initialState() {return new State($\emptyset$,$C$,$P$) }
goal($s$) {
  // the goal is the empty set of distractors
  if $s.C = \emptyset$ then return true
  else return false
}
makeRefExp() {
  // create a initial queue with a single node
  $nodeQueue \leftarrow$ [new Node(initialState())]
  while $nodeQueue \neq \emptyset$ do
    $node \leftarrow$ removeFront($nodeQueue$)
    if goal(node.getState()) then
      return node // success
    end
    $nodeQueue \leftarrow$ queue($nodeQueue$,expand($node$))
  end
  return nil } // failure
```

The first algorithm, we derive from the FB algorithm is one that aims to find the shortest most human like referring expression, which we will call **FBS**. It uses the nearest neighbour metric to select the most human-like among the descriptions that all have the same minimal number of properties in those cases where there are several. The second algorithm, we derive from the FB is encapsulated by a loop which continues to create larger combinations.

The goal of this algorithm is to compute all possible combinations. From a strict point of view, it is not a FB algorithm, since the goals are different. However, since it is based on the FB, we call it **FBN**.

Each of the results of the FBN as well as of the FBS is compared by the nearest neighbour algorithm using the dice metric to each of the referring expression in the training set that contains samples which identify the same entity. The referring expression with the highest score, that means that one which is most similar to one of the referring expressions in the training set, is return as result. For the evaluation, this result referring expression is compared using again the dice metric to that referring expression which was build by a human that is in the test set for exactly the same entity for which the referring expression was automatically build but of course not known by the algorithm.

Definition 2: The Full Brevity Algorithm

```
expand($node$) {
   $N \leftarrow \emptyset$
   $s \leftarrow node.getState()$
   foreach $p \in s.P$ do
      $N \leftarrow N \cup \{$ createNode $(node, p)\}$
   end
   return $N$
}
createNode($node$, $p$) {
   $s \leftarrow node.getState()$
   $out \leftarrow$ rulesOut($p$, $s.C$)
   if $out \neq \emptyset$ & $p \neq$"type" then
      return new Node($s.C - out$, $s.L \cup \{p\}$,
                      $s.P - \{p\}$)
   else return new Node($s.C$, $s.L$, $s.P - \{p\}$)
}
```

Incremental algorithm using C4.5 (**IAC**). Definition 3 shows the Incremental Algorithm as introduced by Dale and Reiter (1995) except that one line is commented out and the two lines after the **if** and before the comment are added. Instead of the predefined list of properties, we use a function to retrieve the next property. This function builds out of the available properties for the intended referent r and the already selected properties $s.L$ an instance which is classified by a decision tree that was build by C4.5[1] from the instances created out of the training set, cf. (Quinlan, 1993). The decision tree gives as result due to the input instance the property which should be selected next. If the chosen property have been already selected then the result is invalid. In this case, as next property, the property with the highest frequency from the remaining properties is chosen. An instance (referent in question) looks like the following two lines:

```
n,-,y,red,n,-n,-,n,-,n,-,n,-,n,-,n-,y,right,n,-,y,large,y,sofa...
n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n
```

The first line defines the properties of the entity for that the referring expression is searched. The `n` or `y` defines, whether a property is present (`y`) or not (`n`) and the values after the `y`'s and `n`'s define the value of that property. For instance the first property is *age* and the second is *colour*. The second line defines, whether a property was already chosen for this referring expression. In the example, none of them were chose, what represents the initial state. In most of the initial states the result of the classification is the property *type*. Training examples have the same structure except that additionally the class is defined. The training examples are build from the training set. Instances are build for each referent accordingly to the number of properties with increasing length using as class the property with the highest frequency. The following training examples are build from a referent with the values *large red sofa*.

```
n,-,y,red,n,-n,-,n,-,n,-,n,-,n,-,n-,y,right,n,-,y,large,y,sofa...
n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,type

n,-,y,red,n,-n,-,n,-,n,-,n,-,n,-,n-,y,right,n,-,y,large,y,sofa...
y,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,colour

n,-,y,red,n,-n,-,n,-,n,-,n,-,n,-,n-,y,right,n,-,y,large,y,sofa...
y,n,y,n,n,n,n,n,n,n,n,n,n,n,n,n,size
```

Definition 3: The Incremental Algorithm

```
expand($node$) {
   $N \leftarrow \emptyset$
   $s \leftarrow node.getState()$
   if $s.P \neq \emptyset$ then
      $p \leftarrow$ askC4.5($s.L$, $r$, $s.P$) for the next property
      if $p = \emptyset$ then select $p \in s.P$ with highest frequency
      // $p \leftarrow$ choose the first p in O, where $p \in s.P$
      $N \leftarrow N \cup \{$ createNode($node, p$) $\}$
   end
   return $N$
}
```

3 Evaluation and Conclusions

Table 1 shows the results for the systems which have been trained on the training data and evaluated against the development data. Section 2 describes the IS-FBS, IS-FBN and IS-IAC. The IAF algorithm was developed as base line, in order to prove that

[1]We tried several other learning techniques like support vector machines, ID3, different types of Bayesian networks, multilayer perceptron, but C4.5 close to SVN performed best.

the learning technique C4.5 for the attribute selection improves the result at all compared to a naive approach which selects the next attribute only due to the frequency as they occur in the training data. The figures show the improvement clearly.

The reason for the development of the FBS was two folded: The FBS algorithm is a base line for the FBN algorithm and it produces the shortest referring expression which is at the same time similar to that produced by humans.

Algorithm	People	Furniture	Avg.
IS-FBS	0.32	0.39	0.355
IS-FBN	0.744	0.800	0.772
IS-IAC	0.696	0.752	0.724
IS-IAF	0.64	0.72	0.68

Table 1: Summary of the results.

Since in some experiments the IAC could beat the FBN, we measure the influence of the size of the training set. We used an n-fold cross validation leaving one out in order to get a high precision for the results. Table 2 shows the results. In the first experiment only the development data was used, in the second experiment only the training data and finally, the training data and the development data was used.

Algorithm	dev	train	train+dev
IS-FBN	0.68	0.74	0.76
IS-IAC	0.72	0.74	0.74

Table 2: Influence of the amount of training data.

The IAC shows better results on smaller training set compared to the FBN. However, the accuracy of the FBN is higher when more training data is available. The difference of the accuracy with 0.02 between this two algorithms is very low. An advantage of the IAC is that it is faster, since it has not to explore a large amount of combinations. Another experiment made a surprising fact about the IAC obvious. During a n-fold cross validation always the complete set to which the test trial belongs has been left out. This had none or no negative effect on the results. The answer gives a look on the decision tree. The decision tree for furniture is shown in Figure 1. The comments added to the decision tree explain the meaning of the tree. The two numbers in the braces at the leafs indicate the training instances which fall into the class. The first number gives the number of correct classified training instances and the second number the number of wrongly classified instances.

```
// **if** the property (attribute) type has been chosen **then**
typea = y
    // **if** colour has been chosen **then**
    coloura = y
        // **if** type has the value sofa **then** chose size
        typev = sofa: size (5.0/2.0)
        // **if** type has the value sofa **then** chose orientation
        typev = chair: orientation (18.0/11.0)
        // etc.
        typev = other: orientation (0.0)
        typev = desk: orientation (6.0/3.0)
        typev = fan: size (3.0)
        typev = -: orientation (0.0)
    coloura = n
        // This part is not completely logic
        // since the x-dimensiona is never chosen
        x-dimensiona = y
            y-dimensiona = y: colour (6.0/1.0)
            y-dimensiona = n: y-dimension (6.0)
        x-dimensiona = n: colour (75.0/12.0)
typea = n: type(122.0/45.0)
```

Figure 1: Decision tree.

The **a** attached to a property indicates the presence or absence of property and the **v** indicates a property with a distinct value. The property after the colon is the property the decision tree suggest to chose next. The decision tree has only 15 nodes and 10 leaves, which is very small.

Humans seem to prefer in case of furniture as provided in the corpus, colour, then somehow the position and then depending on the entity in question the size or orientation. The decision tree for the selection of properties for people is much larger. It contains 63 nodes and 33 leaves. The top level nodes of this tree are the properties *hasGlasses* followed by *hasBeard*. These properties seems to be the most salient properties looking in human faces. Finally, the IAC could not beat the FBN. Nevertheless, the IAC has many advantages, it can be trained on very small training sets, it is fast, and it does not only imitate the referring expressions of humans.

References

B. Bohnet and R. Dale. 2005. Viewing referring expression generation as search. In *IJCAI*, pages 1004–1009.

R. Dale and E. Reiter. 1995. Computational Interpretations of the Gricean Maxims in the Generation of Referring Expressions. *Cognitive Science*, 19(2):233–263.

J. R. Quinlan. 1993. *C4.5 Programs for Machine Learning*. Morgan Kaufmann, California.

H. Simon and A. Newell. 1963. GPS, A Program that Simulates Thought. In *Computers and Thought*, pages 279–293.