

A Best-First Search Algorithm for Generating Referring Expressions

Helmut Horacek

Universität des Saarlandes, FR 6.2 Informatik
Postfach 151150, D-66041 Saarbrücken, Germany
email: horacek@cs.uni-sb.de

Abstract

Existing algorithms for generating referential descriptions to sets of objects have serious deficits: while incremental approaches may produce ambiguous and redundant expressions, exhaustive searches are computationally expensive. Mediating between these extreme control regimes, we propose a best-first searching algorithm for uniquely identifying sets of objects. We incorporate linguistically motivated preferences and several techniques to cut down the search space. Preliminary results show the effectiveness of the new algorithm.

1 Introduction

A *referential description* (Donellan 1966) serves the purpose of letting the addressee identify an object or a set of objects out of the *context set*, the objects assumed to be in the current focus of attention. The referring expression to be generated needs to be a *distinguishing description*, that is, a description of the *intended referent(s)*, the *target set*. Its elements are to be distinguished from *potential distractors* (McDonald 1981), the *contrast set*, which entails all elements of the *context set* except the *intended referent(s)*.

Several algorithms have been developed for this purpose, differing in terms of computational efficiency, quality and coverage. For identifying sets of objects rather than individuals, they typi-

cally suffer from complementary deficits: while incremental approaches may produce ambiguous, redundant expressions, exhaustive searches are computationally expensive. Mediating between these extreme control regimes, we propose a best-first search algorithm for uniquely identifying sets of objects, by incorporating linguistically motivated preferences and techniques to cut down the search space. Preliminary results show the effectiveness of the new algorithm.

This paper is organized as follows. We review relevant work in the field and motivate our goals. Then we describe the new algorithm. Finally, we illustrate its functionality by some examples.

2 Motivation and Previous Approaches

Generating referring expressions has been pursued since the eighties (Appelt 1985, Kronfeld 1986, Appelt and Kronfeld 1987). Later, Dale and Reiter have put considerably more emphasis on computational efficiency in the systems EPICURE (Dale 1988), FN (Reiter 1990), and IDAS (Reiter, Dale 1992). Their algorithms are sensitive to human preferences (e.g., preferring basic categories (Rosch 1978)), and efficient in the sense of minimality of the elements appearing in the resulting referring expression. They differ, however, in terms of the precise interpretation of the minimality criterion. In the mid-nineties, the associated debate of computational efficiency versus minimality of the elements appearing in the resulting referring

expression seemed to be settled in favor of the incremental approach (Dale and Reiter, 1995) – motivated by various results of psychological experiments (see Levelt 1989), certain non-minimal expressions are tolerated in favor of adopting the simple and fast strategy of incrementally selecting ambiguity-reducing attributes from a domain-dependent preference list.

With the extension of the algorithm's scope to the identification of sets of objects rather than individuals (most recently, van Deemter 2002), the incremental strategy was in some sense put to the extreme. Since only few attributes typically apply to all intended referents, boolean combinations of attributes (including negations) are composed into a distinguishing description. This is done by successively collecting single attributes, combinations of two attributes, three attributes, and so on, as long as each of them reduces the set of potential distractors. The a priori preference for structurally simpler combinations constitutes a strong commitment. It may prove unjustified in view of their actual contribution to exclude potential distractors, as demonstrated by Gardent (2002) – see the examples with objects and descriptors as given in Figures 1 and 2. In the first example, x_1 and x_2 are the intended referents. An incremental algorithm would select first the attribute *board-member*, excluding x_6 , then \neg *treasurer*, further excluding x_3 , and only then the disjunction *president* \vee *secretary*. That description could be realized as “a board member, which is the president or the secretary, but not the treasurer”, which is highly redundant compared to “the president or the secretary”. In the second example, $x_5, x_6, x_9,$ and x_{10} are the intended referents. After picking *white* as a descriptor, the incremental algorithm can choose from many alternatives for disjunctions of two attributes. Gardent gives *big* \vee *cow*, *Holstein* \vee

<i>descriptors/objects</i>	x_1	x_2	x_3	x_4	x_5	x_6
president	•					
secretary		•				
treasurer			•			
board-member	•	•	•	•	•	
member	•	•	•	•	•	•

Figure 1. Example 1 taken from (Gardent, 2002)

\neg *small*, \vee *Jersey* \vee \neg *medium* as an intermediate result, potentially verbalized as “the white things that are big or a cow, a Holstein or not small, and a Jersey or not medium”. This is still not a distinguishing description, since it entails x_3 and x_{10} , and this verbalization is much inferior to “the pitbul, the poodle, the Holstein, and the Jersey.” The latter is generated by the constraint-based search developed by Gardent, but it takes 1.4 sec, which is considerable for the small example. In the best-first procedure, we reduce this search space, but we also avoid strong commitments.

3 The Algorithm at a Glance

Basically, the best-first search algorithm is a generalization of the incremental version: instead of successively adding attributes to the full expression generated so far, all intermediate results are accessible for this operation. The “best” among the potential expansion points is determined according to some measure incorporating the complexity of partial descriptions generated so far, the number of potential distractors still to be excluded, and the complexity of descriptors still unused at each state. The expansion process is guided by linguistically motivated preferences (1 and 2, from (Dale, Reiter 1995), adapted to boolean combinations):

1. First, a boolean combination of category descriptors is chosen, other descriptors later. This excludes “mixed” combinations, such as *big* \vee *cow*. Moreover, category descriptors are unconditionally included in the expression, so that not always a minimal description is obtained (excluding, e.g., “the white things”).

<i>descriptors/objects</i>	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}
white	•	•	•	•	•	•	•	•	•	•	•
dog			•						•	•	•
cow				•	•	•	•				
big									•	•	•
small						•	•				
medium-sized				•	•						
pitbul											•
poodle									•		
holstein						•					
jersey						•					

Figure 2. Example 2 taken from (Gardent, 2002)

2. Descriptors are organized in a taxonomy, to capture generalizations; the associated redundancies are exploited in the selection process.
3. Descriptor combinations are limited in size.
4. Negations are penalized (1 point), affecting the ordering of boolean combinations accordingly. E.g, $a \vee b \vee c$ precedes $\neg d \vee \neg e$, but $a \vee b$ precedes $\neg c$ (they are scored as equal).

In addition, efficiency in exploring the search space is greatly supported by two cut-off techniques, termed as *dominance* and *value* cut-offs. A *dominance* cut-off is carried out locally for sibling nodes, when two partial descriptions exclude the same set of potential distractors, and the same set of descriptors is still available. Then the variant which is evaluated worse (in terms of number of descriptors) is discarded. This step is justified by the compositionality in mapping descriptors onto surface expressions, assuming conflation is not possible. A *value* cut-off is carried out globally after a solution has been found. This is done for the nodes whose score of the description generated so far, augmented by the minimal value of the description required for excluding the remaining potential distractors, surpasses the evaluation of the best solution.

4 Formalization of the Algorithm

The algorithm operates on a tree of nodes which are implemented as structured objects, with the following properties, accessible as functions:

- *State*, which is open, closed, final, or cut-off
- *Description*, a boolean descriptor combination
- *Distractors*, remaining objects to be excluded
- *Score*, the quality evaluation of the description
- *Assess*, the likely evaluation for completion
- *Minassess*, the possible minimum for *Assess*
- *Successors*, pointers to daughter nodes
- *Nextprop*, boolean combination for successors

The tree is initialized by a root node with open state, empty description, all distractors, no successors, an empty category as *Nextprop*, and scores according to the evaluation function used. When expanding a node, its successor with a suitable boolean combination of descriptors is created by the function *Create-Successor*, which updates the property *Description* by accumul-

ation and computes the *Distractors* and all evaluation properties accordingly. The property *Nextprop* is the first non-category atomic descriptor for successors of the root node. For successors of interior nodes, it is the descriptor combination following the one chosen at the mother node. The generation of boolean combinations is done by the function *Generate-Next*. It successively builds increasingly complex disjunctions of descriptors and their negation, starting with the one following *Nextprop*, until a combination of limited complexity is found, where:

1. The combination by itself is not redundant
2. It subsumes the target set
3. It further reduces the distractors of the node
4. The reduced set of distractors is not equal to or a superset of the *Distractor* property of a preceding sibling node (*dominance cut-off*)

The best-first search is performed by the procedure *Search* (Figure 3). It maintains the variables

Procedure Search

```

Best ← Root
Bestscore ← ∞; Estimate ← Assess(Best)
1 if State(Best) = Closed
  then return failure endif (1)
if State(Best) = Final
  then return Description(Best) endif (2)
Distractors ← Generate-Next(Best) (3)
if Distractors = nil
  then State(Best) ← Closed
  else Evaluate(Best) (4)
    New ← Create-Successor(Best, Distractors)
    if Distractors(New) = nil then (5)
      State(New) ← Final (6)
      if Bestscore > Score(New) then (7)
        Bestscore ← Score(New)
        for every node n do
          if (State(n) = Open) and (Bestscore
            < (Score(n) + Minassess(n))) (8)
            then State(n) ← Cut-off endif (9)
          next endif endif endif
        for every node n do
          if Assess(n) < Estimate
            then Estimate ← (Score(n) + Assess(n))
            Best ← n endif (10)
        next
      goto Step 1

```

Figure 3. Pseudo-code of the algorithm

- *Best*, the current node under consideration
- *Bestscore*, assessing the best solution found
- *Estimate*, the likely best score of open nodes

The procedure starts with the root node as *Best*. It enters a loop with two termination criteria:

- No more descriptors are available (1)
- A description found is proved to be best (2)

If neither of these is the case, extending *Best* is attempted (3). If unsuccessful, *Best* is closed. Otherwise, it is re-evaluated, and a successor is created (4). If the description associated with this new node excludes all distractors (5), a solution is found (6). If it is better than previously found ones (or the first one) (7), the global score is updated, and all open nodes are tested whether their score can get better than this score (8). The state of such a node is set to *cut-off*, a *value* cut-off (9). Finally, a new *Best* node is chosen (10).

5 Preliminary Results

In the implementation, we have elaborated knowledge bases for the examples in Section 2. In example 1, only three nodes are generated as successors of the root node, representing the descriptions “board member”, “not treasurer”, and “president or secretary”, the last one being the optimal solution. These descriptors are also generated by the incremental algorithm, but as a single expression rather than as alternatives. In example 2, seven nodes are generated, four of them as successors of the root node. They represent the descriptions “dog or cow”, “dog, Jersey or Holstein”, “cow, pitbul or poodle”, and “pitbul, poodle, Jersey or Holstein”, the last one being the optimal solution. The others are extended by “big, medium-sized, or small”, yielding the remaining three nodes. No nodes are generated for the descriptions “not cow, Jersey or Holstein”, and “not dog, pitbul or poodle”, due to *dominance* cut-offs. The program is written in CommonLisp, running on an AMD Athlon processor with 1200 MHz. Computation times are 11 and 400 msec for the two examples, which is 7 resp. 3.5 times faster than the exhaustive search in (Gardent 2002). Hence, using the search restrictions and the representation dependencies cuts down the search space considerably.

6 Conclusion

In this paper, we have presented a best-first search algorithm for producing referring expressions that identify sets of objects. The power of the algorithm comes from linguistically motivated restrictions and preferences, and from a variety of cut-off techniques. Preliminary results show improvements in terms of quality over the incremental algorithm and in terms of speed when compared to exhaustive searches.

References

- Appelt, D. 1985. Planning English Referring Expressions. In *Artificial Intelligence* 26, pp. 1-33.
- Appelt, D., and Kronfeld, A. 1987. A Computational Model of Referring. In Proc. of *IJCAI-87*, pp. 640-647.
- Dale, R. 1988. Generating Referring Expressions in a Domain of Objects and Processes. PhD Thesis, Centre for Cognitive Science, University of Edinburgh.
- Dale, R., and Reiter, E. 1995. Computational Interpretations of the Gricean Maxims in the Generation of Referring Expressions. *Cognitive Science* 18, pp. 233-263.
- Donellan, K. 1966. Reference and Definite Description. *Philosophical Review* 75, pp. 281-304.
- Gardent, C. 2002. Generating Minimal Definite Descriptions. In Proc. of *ACL-2002*, pp. 96-103.
- Kronfeld, A. 1986. Donellan's Distinction and a Computational Model of Reference. In Proc. of *ACL-86*, pp. 186-191.
- Levelt, 1989. *Speaking: From Intention to Articulation*. MIT Press.
- McDonald, D. 1981. Natural Language Generation as a Process of Decision Making under Constraints. PhD thesis, MIT.
- Reiter, E. 1990. Generating Descriptions that Exploit a User's Domain Knowledge. In *Current Issues in Natural Language Generation*, R. Dale, C. Mellish, M. Zock (eds.), pp. 257-285.
- Reiter, E., and Dale, R. 1992. Generating Definite NP Referring Expressions. In Proc. of *COLING-92*.
- Rosch, E. 1978. *Principles of Categorization*. In *Cognition and Categorization*, E. Rosch, B. Lloyd (eds.), pp. 27-48, L. Erlbaum, Hillsdale, New Jersey.
- van Deemter, K. 2002. Generating Referring Expressions: Boolean Extensions of the Incremental Algorithm. *Computational Linguistics*, 28(1), pp. 37-52.