A System for Generating Descriptions of Sets of Objects in a Rich Variety

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

Abstract

Even ambitious algorithms for the generation of referring expressions that identify sets of objects are restricted in terms of efficiency or in their expressive repertoire. In this paper, we report on a system that applies a best-first searching procedure, enhancing both its effectiveness and the variety of expressions it can generate.

1 Introduction

Generating referring expressions has recently been extended from the identification of single to sets of objects. However, existing algorithms suffer in terms of efficiency and expressiveness. In this paper, we report on a system that applies a best-first searching procedure, with an enhanced effectiveness and a larger variety of expressions it can generate. The system's repertoire includes compositions of partially identifying expressions and descriptions of objects to be excluded, thereby taking into account impacts on surface forms.

Throughout this paper, we refer to a scenario with a set of 12 vehicles as defined in Figure 1. All vehicles are identifiable individually, to make the identification task meaningful. Only minor differences hold between some of these vehicles, which makes the identification task challenging.

This paper is organized as follows. First, we motivate our goals. Then we describe techniques for enhancing efficiency. We follow by illustrating improvements of expressiveness. Finally, we evaluate several efficiency-related techniques.

2 Motivation

Identifying sets of objects originally followed the incremental algorithm (Dale and Reiter 1995), as in (Bateman 1999), (Stone 2000) and (Krahmer et al. 2003), with limited coverage, since only few attributes typically apply to all intended referents and to none of the potential distractors. Therefore, van Deemter (2002) has extended the set of descriptors to boolean combinations of attributes, including negations. Unfortunately, when applying the incremental strategy, this may lead to the inclusion of too many redundant descriptors in the final specification. This deficit disappeared using an exhaustive search (Gardent 2002), but run-time then increases considerably. Mediating between these two extreme search paradigms, we have developed a best-first searching algorithm that avoids the major deficit of the incremental approach (Horacek 2003). Since its intermediate results can also be used as partial descriptions, we build on the flexibility of this new algorithm to extend its expressive capabilities. In addition, we further enhance its efficiency-seeking measures.

These extensions attack the deficits previous algorithms share, according to (Horacek 2004):

- Expressions produced may become lengthy: for identifying sets of vehicles in the scenario in Figure 1, we have obtained non-redundant specifications with up to 8 descriptors.
- Specifications may contain some disjunctions, frequently causing the production of structurally ambiguous expressions (Gardent 2002) – "trucks and sportscars which are white or in the center" referring to x_1, x_5, x_{11} (Figure 1).

We avoid these deficits by not restricting boolean expressions to a form with conjunctions as top level operators, as others always do. This allows us to incorporate descriptions of objects to be excluded, to produce enumerations and compositions of descriptions of subsets of the intended referents, and to build compositions of increasingly restricting descriptions of these referents.

	Objects												
Descriptors	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}
vehicle		•	•	•	•	•	•	•	•	•	•	•	•
car				•	•	•	•			•	•	•	•
sportscar						•	•					•	•
truck		•	•					•	•				
blue			•							•			•
red					•		•	•	•		•	•	
white		•		•		•							
center					•	•				•		•	
left			•						•		•		•
right		•		•			•	•					
big		•	•	•							•	•	•
small					•	•	•	•	•	•			
new		•			•	•			•		•		•
old			•	•			•	•		•		•	

Figure 1. Example scenario with 12 vehicles

3 The Best-First Procedure

The basic mechanism of 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, producing an *optimal* solution, if completed – see (Horacek 2003) for details. This algorithm uses two cut-off techniques, assuming conflation (e.g., the descriptors *man* and *unmarried* can be verbalized as "bachelor") is not possible:

- A *dominance* cut-off is carried out locally for sibling nodes, when two partial descriptions exclude the same set of potential distractors, the same set of descriptors still being available. The variant evaluated worse is discarded.
- A *value* cut-off is carried out globally after a solution has been found. It is done for nodes whose most optimistic evaluation (including the minimal value of the description required for excluding the remaining potential distractors), surpasses the evaluation of that solution.

Applying any of these cut-offs only serves to gain speed and does not change the final result.

3.1 Efficiency-Enhancing Measures

We have enhanced this repertoire by a *complexity* cut-off, carried out prior to further expanding a node if the boolean combination of descriptors build leads to a description that is more complex than a given threshold. For this threshold, we use the complexity of descriptions identifying each referent individually, which is an enumeration.

The generation of boolean combinations is a critical part of the algorithm, since it is its most time-consuming component. Redundancies must be avoided, which requires more effort than previous approaches due to our hierarchical organization of property values. This burden is split between a static representation of implications, compiled from the underlying knowledge base about specializations, and the function *Generate-Next*, which accesses these data. Four implications hold between properties and their negations:

implies (p,q) if *specializes*(p,q) holds
implies (p,¬q) if *incompatible*(p,q) holds
implies (¬p,q) if *opposite*(p,q) holds
implies (¬p,¬q) if *generalizes*(p,q) holds

Then the predicates *subsumes* and *redundant* can be defined for properties (or their negations):

subsumes(p,q) ≡ *implies* (q,p)
redundant(p,q) ≡ ¬(*subsumes*(p,q) ∨ *subsumes*(q,p))

The function *Generate-Next* (Figure 1) successively builds increasingly complex disjunctions of descriptors and their negation. To start with, the procedure *Increment* produces the next property combination with given complexity, if existing (1). Otherwise (2), that complexity is augmented (9) before generating the next combination, unless the complexity limit is reached (8), causing a *complexity cut-off*. For a property combination, it is tested whether all its properties are pairwise redundant (3), then the next combination is built. If a non-redundant combination is found, it must pass the following tests:

1. It subsumes the target set (4).
2. It further reduces the set of distractors (5).
3. The reduced set of distractors is not equal to or a superset of the distractor associated with a sibling node already created; otherwise, a *dominance cut-off* applies (6).

If successful, that combination is returned, otherwise building combinations is resumed (7).

3.2 Enhancing the Best-First Procedure

We have incorporated a number of improvements over the original version of the procedure:

- Treating linguistically motivated preferences as options rather than restrictions
- Putting limitations on the complexity of specifications, to control comprehensibility
- Enhancing the expressive repertoire by descriptions of subsets of referents and by descriptions of referents to be excluded
- Producing a sequence of increasingly restricting descriptions rather than a single one.

Procedure Generate-Next(*Current-Prop-Comb*)

1 *Nextprop* ← Increment(*Current-Prop-Comb*) (1)
if *Nextprop* = nil **then goto** Step 2 **endif** (2)
if redundant(*p*,*q*) for any *p*,*q* ∈ *Nextprop* (3)
then goto Step 1 **endif**
if subsumes(*Nextprop*,Properties-of*(T)*)
for all *T* ∈ *Target* **and** (4)
¬subsumes(*Nextprop*,Props*(D))*
for some *D* ∈ *Distractors(Best-Node)* (5)
and ¬ *Q* ⊇ *R*, where
R = {subsumes(Properties-of(*P),Nextprop*)},
Q ={subsumes(Properties-of(*P),*
Description(*N*))}
for all *P* ∈ *Distractors*, (6)
some *N* ∈ successor(*Best-Node*)
then return *Nextprop* (7)
else goto Step 1 **endif** (*Dominance cut-off*)
2 **if** (Score(*Description(Best-Node*)) +
Score(*Nextprop*)) ≥ *Complexity-limit* (8)
then return nil (*Complexity cut-off*)
else *Nextprop* ← Increment-size(*Nextprop*)
goto Step 1 **endif** (9)

Figure 2. Pseudo-code of descriptor generation

In the following, we summarize each of these (see (Horacek 2004) for details).

The following linguistically motivated preferences are treated as options: a boolean combination of descriptors that express the category of the object (by a head noun) is chosen first, other (attribute) descriptors later, since a category must be chosen anyway. Moreover, we reduce the set of potential solutions by excluding "mixed" boolean combinations, that is disjunctions of a category and attributes, such as $car \vee red$, which are unnatural and awkward to express verbally.

To strengthen comprehensibility, we specify limitations on the surface form of descriptions, including places for the head noun, pre- and postnominal modifiers, and relative clauses. Maximum numbers for each of these positions can be given, also specifying places as alternative ones, thus limiting the number of components in conjoined expressions. By associating descriptors with surface positions they can take, these specifications allow one to control the surface structure of the descriptions during searching.

For partial descriptions with multiple disjunctions, recasting the expression built as a partial description is attempted to remain within given limits. These descriptions are always of the form $\wedge_{i=1,n} (\vee_{j=1,mi} P_{ij})$, where each P_{ij} is a positive or negative descriptor. Even in moderately complex instances of this conjoined expression, several elements may consist of disjunctions of more than one descriptor. In such a constellation,we pick up one disjunction, for example $\vee_{j=1,mk} P_{kj}$ for some k, transforming that expression by applying distributivity. This amounts to partitioning the set of intended referents into subsets, where each of the components of the new top level disjunction describes one of these subsets. Consider, for example, "the sportscars that are not red and the small trucks" identifying x_5, x_7, x_8, and x_{12} in two components rather than by the involved one-shot "the vehicles that are a sportscar or small, and either a truck or not red." In addition, descriptions may specify *exceptions*: describing some of the referents to be excluded may lead to shorter expressions than expanding the description of the intended referents, so that we integrate it in the expressive repertoire – for example, "the vehicles on the right, but not the red truck", identifying x_1, x_3, and x_6 by excluding x_7 in the locally restricted context.

In accordance with these specifications, the best-first search is invoked to produce an identifying description. This may not always be possible in complex situations. If this is the case, the best partial solution is taken, and the search is repeated within the restricted context defined by the descriptions generated so far. By this procedure, a sequence of descriptions is generated rather than a single one. Consider, for example, "one of the trucks and the sportscars, all not white. The truck stands on the right", identifying x_6, x_7, x_{11} and x_{12} out of all 12 vehicles (in Figure 1) in two passes.

3.3 An Example

We illustrate the behavior of the system by a small example. Let $\{x_1, x_3, x_6\}$ in Figure 1 be the set of intended referents. Specifications for maximum complexity of surface forms allow head nouns, pre- and postnominal modifiers, at most one of them as a conjoined expression, and a relative clause or a "but"-modifier expressing an exception. Only two descriptors apply to all intended referents, *vehicle* and *right*. Even if *vehicle* is chosen first, subsequent searching only expands on the partial description with *right*, since it excludes a superset of the objects *vehicle* does: only x_7 is remaining. The next simplest descriptor combination is $car \vee white$, which would allow complete identification of the intended referents. Since it can only be expressed by a relative clause, for which conjoined expressions are not allowed, recasting the description is attempted. This yields $(car \wedge right) \vee (white \wedge right)$, which is a possible solution. Since a head noun is required for the second part, adding a further descriptor, an attempt is made to improve the solution, through finding an alternative to $car \vee white$. Describing the complement constitutes such an alternative, since identification is required for x_7 only. This can be done by selecting *truck* and, afterwards, any of the descriptors *red*, *small*, and *old* (let us say, we pick *red*). This yields $right \wedge \neg (truck \wedge red)$ as an alternative solution, with *vehicle* being added to obtain a head noun. Altogether, a surface generator could then generate "the vehicles on the right, but not the red truck ", resp. "the cars and the white vehicle, both on the right" – the latter with a clever aggregation module.

4 Experimental Results

We have implemented the algoritm in Common Lisp, on an Intel Pentium processor with 2600 MHz. In the following elaborations, we use natural language descriptions for reasons of readability, even though our algorithm only produces boolean combinations of descriptors.

We evaluate our algorithm from three perspectives: 1) effects of the linguistically motivated restrictions, 2) effectiveness of the cut-off techniques, and 3) the behavior in scaling up for larger examples. For this purpose, we have built all subsets of two, three, and four vehicles, out of the vehicles x_1 to x_6, which yields 50 cases.

In order to test the effects of the linguistically motivated reductions, we have used two versions

cut-offs (v=value, d=dominance, c=complexity)	v&d&c	v&c	d&c	c	v&d	d	v
				time (msec)			
minimum	10	10	60	90	10	90	10
maximum	690	1150	1910	19210	1100	4550	2320
average	121.5	131.6	354.8	1133.1	140.5	595.0	168.1
				tree size (nodes)			
maximum	9	71	11	945	9	11	71
average	2.2	3.86	2.33	61.64	2.2	2.33	3.88

Table 1. Searches comparing effects of cut-offs

nr. of distractors	6	7	8	9	10	12	15	20	25
					time (msec)				
minimum	10	10	10	10	10	30	60	100	120
maximum	490	2300	3880	4100	4430	6530	53390	88120	141200
average	116	282	417	484	705	1120	5366	12325	24838
					max nr. of				
tree nodes	9	10	12	16	27	61	106	303	907
descriptors	5	5	5	5	5	5	6	8	10

Table 2. Searches with varying sets of distractors

of the 50 cases, one with all properties, and one without size and age. In these runs, the maximum number of descriptors chosen was 5, and search trees grew up to 9 with and 20 nodes without using the linguistically motivated reductions. The average search times were 127.7 resp. 440.5 msec, with a maximum of 950 resp. 2590 msec.

In order to compare the effectiveness of the cut-off techniques, we have run the same sample of 100 cases (50 with and 50 without size and age), with all combinations of at least one cut-off technique. Table 1 illustrates the results. Among others, they demonstrate that search times are not proportional to tree sizes, since a lot of effort is devoted to justify the avoidance of expansions, which varies among cut-off techniques. It turns out that the *value* cut-off is the most effective one, which underpins the importance of finding a solution quickly. Looking at individual examples reveals that the complementary effects of *dominance* and *complexity* cut-offs are significant only for examples with larger solutions.

Finally, we have tested the algorithm's scalability, by increasing the number of distractors, with up to 25 vehicles (similar to x_1 to x_{12}, but distinct from one another). The same 100 cases have been used as before, with all cut-off criteria. The results appear in Table 2. They demonstrate that the problem tends to get unmanagable for more than 12 distractors in both search time and number of descriptors needed for identification, the latter being the reason for the former. However, descriptions consisting of up to 10 descriptors are unlikely to be understandable for humans, anyway – consider, for example, "the cars which are not blue, are old or stand in the center, are new or stand on the right side, are big or not white, and are small or not red" (108110 msec, identifying x_3, x_4, and x_6 out of 25 vehicles). For such complicated cases, identifying objects is broken down into simpler tasks (see Section 3.2). Conversely, useful results may be obtained for a large number of distractors – for example, "the old cars on the right side" (120 msec, identifying x_3 and x_6 out of 25 vehicles).

5 Conclusion

We have presented a system that can produce referring expressions for identifying sets of objects. It has a number of exceptional features, including several efficiency-enhancing measures, the incorporation of exclusion descriptions, and partitioning the identification task into subtasks. The results show that our system has an increased repertoire compared to its predecessors, and it can compute these expressions reasonably fast.

References

John Bateman 1999. Using Aggregation for Selecting Content when Generating Referring Expressions. In Proc. of *37th Annual Meeting of the Association for Computational Linguistics (ACL'99)*, pp. 127-134.

Robert Dale and Ehud Reiter 1995. Computational Interpretations of the Gricean Maxims in the Generation of Referring Expressions. *Cognitive Science* 18: 233-363.

Claire Gardent 2002. Generating Minimal Definite Descriptions. In Proc. of *40th Annual Meeting of the Association for Computational Linguistics (ACL'2002)*, pp. 96-103.

Helmut Horacek 2003. A Best-First Search Algorithm for Generating Referring Expressions. In Proc. of *10th Conference of The European Chapter of the Association for Computational Linguistics (EACL'2003)*, short paper, pp. 103-106.

Helmut Horacek 2004. On Referring to Sets of Objects Naturally. In Proc. of *Third International Natural Language Generation Conference* (*INLG-2004)*.

Emiel Krahmer, Sebastiaan van Erk, and André Verleg 2003. Graph-Based Generation of Referring Expressions. *Computational Linguistics*, 29(1):53-72.

Matthew Stone 2000. On Identifying Sets. In Proc. of *First International Natural Language Generation Conference* (*INLG-2000)*, pp. 116-123.

Kees van Deemter 2002. Generating Referring Expressions: Boolean Extensions of the Incremental Algorithm. *Computational Linguistics*, 28(1):37-52.