Algorithms for Generating Referring Expressions: Do They Do What People Do?

Jette Viethen
Centre for Language Technology
Macquarie University
Sydney NSW 2109
jviethen@ics.mq.edu.au

Robert Dale
Centre for Language Technology
Macquarie University
Sydney NSW 2109
robert.dale@mq.edu.au

Abstract

The natural language generation literature provides many algorithms for the generation of referring expressions. In this paper, we explore the question of whether these algorithms actually produce the kinds of expressions that people produce. We compare the output of three existing algorithms against a data set consisting of human-generated referring expressions, and identify a number of significant differences between what people do and what these algorithms do. On the basis of these observations, we suggest some ways forward that attempt to address these differences.

1 Introduction

The generation of referring expressions (henceforth GRE) — that is, the process of working out what properties of an entity should be used to describe it in such a way as to distinguish it from other entities in the context — is a recurrent theme in the natural language generation literature. The task is discussed informally in some of the earliest work on NLG (in particular, see (Winograd, 1972; McDonald, 1980; Appelt, 1981)), but the first formally explicit algorithm was introduced in (Dale, 1989); this algorithm, often referred to as the Full Brevity (FB) algorithm, has served as a starting point for many subsequent GRE algorithms. To overcome its limitation to one-place predicates, Dale and Haddock (1991) introduced a constraint-based procedure that could generate referring expressions involving relations; and as a response to the computational complexity of 'greedy' algorithms like FB, Reiter and Dale (Reiter and Dale, 1992; Dale and Reiter, 1995) introduced the psycholinguistically motivated Incremental Algorithm (IA). In recent years there have been a number of important extensions to the IA. The Context-Sensitive extension (Krahmer and Theune, 2002) is able to generate referring expressions for the most salient entity in a context; the Boolean Expressions algorithm (van Deemter, 2002) is able to derive expressions containing boolean operators, as in *the cup that does not have a handle*; and the Sets algorithm (van Deemter, 2002) extends the basic approach to references to sets, as in *the red cups*. Some approaches reuse parts of other algorithms: the Branch and Bound algorithm (Krahmer et al., 2003) uses the Full Brevity algorithm, but is able to generate referring expressions with both attributes and relational descriptions using a graph-based technique. There are many other algorithms described in the literature: see, for example, (Horacek, 1997; Bateman, 1999; Stone, 2000; Gardent, 2002). Their general aim is to produce naturalistic referring expressions, often explicitly by means of an attempt to follow the same kinds of principles that we believe people might be following when they produce language — such as the Gricean maxims (Grice, 1975). However, the algorithms have rarely been tested against real data from human referring expression generation.[1]

In this paper, we present a data set containing human-produced referring expressions in a limited domain. Focussing specifically on the algorithms

[1] The only exceptions we know of to this deficit are not directly concerned with the kinds of properties people select, but with phenomena such as how people group entities together (Funakoshi et al., 2004; Gatt, 2006), or with multimodal referring expressions where the linguistic part is not necessarily distinguishing by itself (van der Sluis and Krahmer, 2004).

presented in (Dale, 1989), (Dale and Haddock, 1991) and (Reiter and Dale, 1992), we explore how well these algorithms perform in the same context. There are significant differences between the referring expressions produced by humans, and those produced by the algorithms; we explore these differences and consider what it means for work in the generation of referring expressions.

The remainder of this paper is structured as follows. In Section 2, we introduce the data set of human-produced referring expressions we use; in Section 3, we introduce the representational framework we use to model the domain underlying this data; in Section 4 we introduce the three algorithms considered in this paper; in Section 5 we discuss the results of using these algorithms on the data that represents the model of our domain; in Section 6 we discuss the differences between the output of the algorithms and the human-produced data; and in Section 7 we draw some conclusions and suggest some steps towards addressing the issues we have identified.

2 The Data

Our human-produced referring expressions are drawn from a physical experimental setting consisting of four filing cabinets, each of which is four drawers high, located in a fairly typical academic office. The cabinets are positioned directly next to each other, so that the drawers form a four-by-four grid; each drawer is labelled with a number between 1 and 16 and is coloured either blue, pink, yellow, or orange. There are four drawers of each colour which are distributed randomly over the grid, as shown in Figure 1.

Subjects were given a randomly generated number between 1 and 16, and asked to produce a description of the numbered drawer using any properties other than the number. There were 20 participants in the experiment, resulting in a total of 140 referring expressions. Here are some examples of the referring expressions produced:

(1) the top drawer second from the right [d_3]

(2) the orange drawer on the left [d_9]

(3) the orange drawer between two pink ones [d_{12}]

(4) the bottom left drawer [d_{16}]

Since the selection of which drawer to describe was random, we do not have an equal number of descriptions of each drawer; in fact, the data set ranges from two descriptions of Drawer 1 to 12 descriptions of Drawer 16. One of the most obvious things about the data set is that even the same person may refer to the same entity in different ways on different occasions, with the differences being semantic as well as syntactic.

1 (blue)	2 (orange)	3 (pink)	4 (yellow)
8 (blue)	7 (blue)	6 (yellow)	5 (pink)
9 (orange)	10 (blue)	11 (yellow)	12 (orange)
16 (yellow)	15 (pink)	14 (orange)	13 (pink)

Figure 1: The filing cabinets

We are interested in comparing how algorithms for referring expression generation differ in their outputs from what people do; since these algorithms produce distinguishing descriptions, we therefore removed from the data set 22 descriptions which were ambiguous or referred to a set of drawers. This resulted in a total of 118 distinct referring expressions, with an average of 7.375 distinct referring expressions per drawer.

As the algorithms under scrutiny here are not concerned with the final syntactic realisation of the referring expression produced, we also normalised the human-produced data to remove superficial variations such as the distinction between relative clauses and reduced relatives, and between different lexical items that were synonymous in context, such as *column* and *cabinet*.

Four absolute properties used for describing the drawers can be identified in the natural data produced by the human participants. These are the colour of the drawer; its row and column; and in those cases where the drawer is situated in one of the corners of the grid, its cornerhood.[2] A number of the natural descriptions also made use of the

[2] A question we will return to below is that of how we decide whether to view a particular property as a one-place predicate or as a relation.

Property	Count	% (out of possible)
Row	95	79.66% (118)
Column	88	73.73% (118)
Colour	63	53.39% (118)
Corner	11	40.74% (27)
Relation	15	12.71% (118)

Table 1: The properties used in descriptions

following relational properties that hold between drawers: above, below, next to, right of, left of and between. In Table 1, Count shows the number of descriptions using each property, and the percentages show the ratio of the number of descriptions using each property to the number of descriptions for drawers that possess this property (hence, only 27 of the descriptions referred to corner drawers). We have combined all uses of relations into one row in this table to save space, since, interestingly, their overall use is far below that of the other properties: 103 descriptions (87.3%) did not use relations.

Most algorithms in the literature aim at generating descriptions that are as short as possible, but will under certain circumstances produce redundancy. Some authors, for example (van Deemter and Halldórsson, 2001), have suggested that human-produced descriptions are often not minimal, and this is an intuition that we would generally agree with. However, a strong tendency towards minimality is evident in the human-produced data here: only 29 out of 118 descriptions (24.6%) contain redundant information. Here are a few examples:

- *the yellow drawer in the third column from the left second from the top* [d_6]
- *the blue drawer in the top left corner* [d_1]
- *the orange drawer below the two yellow drawers* [d_{14}]

In the first case, either the colour or column properties are redundant; in the second, colour and corner, or only the grid information, would have been sufficient; and in the third, it would have been sufficient to mention one of the two yellow drawers.

3 Knowledge Representation

In order to use an algorithm to generate referring expressions in this domain, we must first decide how to represent the domain. It turns out that this raises some interesting questions.

We use the symbols $\{d_1, d_2 \ldots d_{16}\}$ as our unique identifying labels for the 16 drawers. Given some d_i, the goal of any given algorithm is then to produce a distinguishing description of that entity with respect to a context consisting of the other 15 drawers.

As is usual, we represent the properties of the domain in terms of attribute–value pairs. Thus we have, for example:

- d_2: ⟨colour, orange⟩, ⟨row, 1⟩, ⟨column, 2⟩, ⟨right-of, d_1⟩, ⟨left-of, d_3⟩, ⟨next-to, d_1⟩, ⟨next-to, d_3⟩, ⟨above, d_7⟩

This drawer is in the top row, so it does not have a property of the form ⟨below, d_2⟩.

The four corner drawers additionally possess the property ⟨position, corner⟩. Cornerhood can be inferred from the row and column information; however, we added this property explicitly because several of the natural descriptions use the property of cornerhood, and it seems plausible that this is a particularly salient property in its own right.

This raises the question of what properties should be encoded explicitly, and which should be inferred. Note that in the example above, we explicitly encode relational properties that could be computed from others, such as left-of and right-of. Since none of the algorithms explored here uses inference over knowledge base properties, we opted here to 'level the playing field' to enable fairer comparison between human-produced and machine-produced descriptions.

A similar question of the role of inference arises with regard to the transitivity of spatial relations. For example, if d_1 is above d_9 and d_9 is above d_{16} , then it can be inferred that d_1 is transitively above d_{16}. In a more complex domain, the implementation of this kind of knowledge might play an important role in generating usful referring expressions. However, the uniformity of our domain results in this inferred knowledge about transitive relations being of little use; in fact, in most cases, the implementation of transitive inference might even result in the generation of unnatural descriptions, such as *the orange drawer (two) right of the blue drawer* for d_{12}.

Another aspect of the representation of relations that requires a decision is that of generalisation:

next-to is a generalisation of the relations left-of and right-of. The only algorithm of those we examine here that provides a mechanism for exploring a generalisation hierarchy is the Incremental Algorithm (Reiter and Dale, 1992), and this cannot handle relations; so, we take the shortcut of explicitly representing the next-to relation for every left-of and right-of relation in the knowledge base. We then implement special-case handling that ensures that, if one of these facts is used, the more general or more specific case is also deleted from the set of properties still available for the description.[3]

4 The Algorithms

As we have already noted above, there is a considerable literature on the generation of referring expressions, and many papers in the area provide detailed algorithms. We focus here on the following algorithms:

- The Full Brevity algorithm (Dale, 1989) attempts to build a minimal distinguishing description by always selecting the most discriminatory property available; see Algorithm 1.

Let L be the set of properties to be realised in our description; let P be the set of properties known to be true of our intended referent r (we assume that P is non-empty); and let C be the set of distractors (the contrast set). The initial conditions are thus as follows:

- $C = \{\langle$*all distractors*$\rangle\}$;
- $P = \{\langle$*all properties true of r*$\rangle\}$;
- $L = \{\}$

In order to describe the intended referent r with respect to the contrast set C, we do the following:

1. Check Success:
   if $|C| = 0$ **then** return L as a distinguishing description
   elseif $P = \emptyset$ **then** fail
   else goto Step 2.
2. Choose Property:
   for each $p_i \in P$ **do**:
   $C_i \leftarrow C \cap \{x | p_i(x)\}$
   Chosen property is p_j, where C_j is the smallest set.
   goto Step 3.
3. Extend Description (wrt the chosen p_j):
   $L \leftarrow L \cup \{p_j\}$
   $C \leftarrow C_j$
   $P \leftarrow P - \{p_j\}$
   goto Step 1.

Algorithm 1: The Full Brevity Algorithm

[3] This is essentially a hack; however, there is clearly a need for some mechanism for handling what we might think of as equivalence classes of properties, and this is effectively a simple approach to this question.

---

1. Check Success
   if Stack is empty **then** return L as a DD
   elseif $|C_v| = 1$ **then** pop Stack & **goto** Step 1
   elseif $P_r = \emptyset$ **then** fail
   else goto Step 2
2. Choose Property
   for each property $p_i \in P_r$ **do**
   $p_i' \leftarrow [r \backslash v]p_i$
   $N_i \leftarrow N \oplus p_i'$
   Chosen prediction is p_j, where N_j contains the smallest set C_v for v.
   goto Step 3
3. Extend Description (w.r.t the chosen p)
   $P_r \leftarrow P_r - \{p\}$
   $p \leftarrow [r \backslash v]p$
   for every other constant r' in p **do**
   associate r' with a new, unique variable v'
   $p \leftarrow [r' \backslash v']p$
   push `Describe`(r',v') onto `Stack`
   initialise a set P_r' of facts true of r'
   $N \leftarrow N \oplus p$
   goto Step 1

Algorithm 2: The Relational Algorithm

MakeReferringExpression(r, C, P) $L \leftarrow \{\}$
for each member A_i of list P **do**
  V = FindBestValue(r, A_i, *BasicLevelValue*(r, A_i))
  if RulesOut($\langle A_i, V\rangle$) $\neq$ nil
  then $L \leftarrow L \cup \{\langle A_i, V\rangle\}$
    $C \leftarrow C -$ RulesOut($\langle A_i, V\rangle$)
  endif
  if $C = \{\}$ **then**
    if $\langle$type, $X\rangle \in L$ for some X
      then return L
      else return $L \cup \{\langle$type, BasicLevelValue(r, type)$\rangle\}$
    endif
  endif
return failure

FindBestValue(r, A, *initial-value*)
if *UserKnows*(r, $\langle A$, *initial-value*$\rangle$) = **true**
then *value* $\leftarrow$ *initial-value*
else *value* $\leftarrow$ no-value
endif
if (*more-specific-value* $\leftarrow$ ***MoreSpecificValue***(**r, A,** *value*)) $\neq$ nil $\wedge$
  (*new-value* $\leftarrow$ ***FindBestValue***(**A,** *more-specific-value*)) $\neq$ nil $\wedge$
  (|RulesOut($\langle A$, *new-value*$\rangle$)| $>$ |RulesOut($\langle A$, *value*$\rangle$)|)
then *value* $\leftarrow$ *new-value*
endif
return *value*

RulesOut($\langle A, V\rangle$)
if V = no-value
then return nil
else return $\{x : x \in C \wedge$ ***UserKnows***(**x,** $\langle A, V\rangle$) = **false**$\}$
endif

Algorithm 3: The Incremental Algorithm

- The relational algorithm from (Dale and Haddock, 1991) uses constraint satisfaction to incorporate relational properties while avoiding infinite regress; see Algorithm 2.

- the Incremental Algorithm (Reiter and Dale, 1992; Dale and Reiter, 1995) considers the available properties to be used in a description via a preference ordering over those properties; see Algorithm 3.

For the purpose of this study, the algorithms were implemented in Common LISP. The mechanism described in (Dale and Reiter, 1995) to handle generalisation hierarchies for values for the different properties, referred to in the algorithm here as FindBestValue, was not implemented since, as discussed earlier, our representation of the domain does not make use of a hierarchy of properties.

5 The Output of the Algorithms

Using the knowledge base described in Section 3, we applied the algorithms from the previous section to see whether the referring expressions they produced were the same as, or similar to, those produced by the human subjects. This quickly gave rise to some situations not explicitly addressed by some of the algorithms; we discuss these in Section 5.1 below. Section 5.2 discusses the extent to which the behaviour of the algorithms matched that of the human data.

5.1 Preference Orderings

The Incremental Algorithm explicitly encodes a preference ordering over the available properties, in an attempt to model what appear to be semi-conventionalised strategies for description that people use. This also has the consequence of avoiding a problem that faces the other two algorithms: since the Full Brevity Algorithm and the Relational Algorithm choose the most discriminatory property at each step, they have to deal with the case where several properties are equally discriminatory. This turns out to be a common situation in our domain. Both algorithms implicitly assume that the choice will be made randomly in these cases; however, it seems to us more natural to control this process by imposing some selection strategy. We do this here by borrowing the idea of preference ordering from the Incremental Algorithm, and using it as a tie-breaker when multiple properties are equally discriminatory.

Not including type information (i.e., the fact that some d_i is a drawer), which has no discriminatory power and therefore will never be chosen by any of the algorithms,[4] there are only four different properties available for the Full Brevity Algorithm and the Incremental Algorithm: row, column, colour, and position. This gives us 4! = 24 different possible preference orderings. Since some of the human-produced descriptions use all four properties, we tested these two algorithms with all 24 preference orderings.

For the Relational Algorithm, we added the five relations next to, left of, right of, above, and below. This results in 9! = 362,880 possible preference orderings; far too many to test. Since we are primarily interested in whether the algorithm can generate the human-produced descriptions, we restricted our testing to those preference orderings that started with a permutation of the properties used by the participants; in addition to the 24 preference orderings above, there are 12 preference orderings that incorporate the relational properties.

5.2 Coverage of the Human Data

Overall, the Full Brevity Algorithm is able to generate 82 out of the 103 non-relational descriptions from the natural data, providing a recall of 79.6%. The recall score for the Incremental Algorithm is 95.1%, generating 98 of the 103 descriptions. As these algorithms do not attempt to generate relational descriptions, the relational data is not taken into account in evaluating the performance here.

Both algorithms are able to generate all the non-relational minimal descriptions found in the human-produced data. The Full Brevity Algorithm unintentionally replicates the redundancy found in nine descriptions, and the Incremental Algorithm produces all but five of the 29 redundant descriptions.

Perhaps surprisingly, the Relational Algorithm does not generate *any* of the human-produced descriptions. We will return to consider why this is the case in the next section.

6 Discussion

There are two significant differences to be considered here: first, the coverage of redundant descriptions by the Full Brevity and Incremental Algo-

[4] Consistent with much other work in the field, we assume that the head noun will always be added irrespective of whether it has any discriminatory power.

rithms; and second, the inability of the Relational Algorithm to replicate any of the human data.

6.1 Coverage of Redundancy

Neither the Full Brevity Algorithm nor the Incremental Algorithm presumes to be able to generate relational descriptions; however, both algorithms are able to produce each of the minimal descriptions from the set of natural data with at least one of the preference orderings. Both also generate several of the redundant descriptions in the natural data set, but do not capture all of the human-generated redundancies.

The Full Brevity Algorithm has as a primary goal the avoidance of redundant descriptions, so it is a sign of the algorithm being consistent with its specification that it covers fewer of the redundant expressions than the Incremental Algorithm. On the other hand, the fact that it produces *any* redundant descriptions signals that the algorithm doesn't quite meet its specification. The cases where the Full Brevity Algorithm produces redundancy are when an entity shares with another entity at least two property-values and, after choosing one of these properties, the next property to be considered is the other shared one, since it has the same or a higher discriminatory power than all other properties. This is a situation that was not considered in the original algorithm; it is related to the problem of what to do when two properties have the same discriminatory power, as noted earlier. In our domain, the situation arises for corner drawers with the same colour (d_4 and d_{16}), and drawers that are not in a corner but for which there is another drawer of the same colour in each of the same row and column (d_7 and d_8).

The Incremental Algorithm, on the other hand, generates redundancy when an object shares at least two property-values with another object and the two shared properties are the first to be considered in the preference ordering. This is possible for corner drawers with the same colour (d_4 and d_{16}) and for drawers for which there is another drawer of the same colour in either the same row, the same column, or both (d_5, d_6, d_7, d_8, d_{10}, d_{11}, d_{13}, d_{15}).

In these terms, the Incremental Algorithm is clearly a better model of the human behaviour than the Full Brevity Algorithm. However, we may ask why the algorithm does not cover all the redundancy found in the human descriptions. The redundant descriptions which the algorithm does not generate are as follows:

(5) *the blue drawer in the top left corner* [d_1]

(6) *the yellow drawer in the top right corner* [d_4]

(7) *the pink drawer in the top of the column second from the right* [d_3]

(8) *the orange drawer in the bottom second from the right* [d_{14}]

(9) *the orange drawer in the bottom of the second column from the right* [d_{14}]

The Incremental Algorithm stops selecting properties when a distinguishing description has been constructed. In Example (6), for example, the algorithm would select any of the following, depending on the preference ordering used:

(10) *the yellow drawer in the corner*

(11) *the top left yellow drawer*

(12) *the drawer in the top left corner*

The human subject, however, has added information beyond what is required. This could be explained by our modelling of cornerhood: in Examples (5) and (6), one has the intuition that the noun *corner* is being added simply to provide a nominal head to the prepositional phrase in an incrementally-constructed expression of the form *the blue drawer in the top right* ..., in much the same way as the head noun *drawer* is added, whereas we have treated it as a distinct property that adds discriminatory power. This again emphasises the important role the underlying representation plays in the generation of referring expressions: if we want to emulate what people do, then we not only need to design algorithms which mirror their behaviour, but these algorithms have to operate over the same kind of data.

6.2 Relational Descriptions

The fact that the Relational Algorithm generates none of the human-generated descriptions is quite disturbing. On closer examination, it transpires that this is because, in this domain, the discriminatory power of relational properties is generally always greater than that of any other property, so a relational property is chosen first. As noted earlier, relational properties appear to be dispreferred

in the human data, so the Relational Algorithm is already disadvantaged. The relatively poor performance of the algorithm is then compounded by its insistence on continuing to use relational properties: an absolute property will only be chosen when either the currently described drawer has no unused relational properties left, or the number of distractors has been reduced so much that the discriminatory power of all remaining relational properties is lower than that of the absolute property, or the absolute property has the same discriminatory power as the best relational one and the absolute property appears before all relations in the preference ordering.

Consequently, whereas a typical human description of drawer d_2 would be *the orange drawer above the blue drawer*, the Relational Algorithm will produce the description *the drawer above the drawer above the drawer above the drawer above the pink drawer*. Not only are there no descriptions of this form in the human-produced data set, but they also sound more like riddles someone might create to intentionally make it hard for the hearer to figure out what is meant.

There are a variety of ways in which the behaviour of this algorithm might be repaired. We are currently exploring whether Krahmer et al's (2003) graph-based approach to GRE is able to provide a better coverage of the data: this algorithm provides the ability to make use of different search strategies and weighting mechanisms when adding properties to a description, and such a mechanism might be used, for example, to counterbalance the Relational Algorithm's heavy bias towards the relations in this domain.

7 Conclusions and Future Work

We have noted a number of regards in which the algorithms we have explored here do not produce outputs that are the same as those produced by humans. Some comments on the generalisability of these results are appropriate.

First, our results may be idiosyncratic to the specifics of the particular domain of our experiment. We would point out, however, that the domain is more complex, and arguably more realistic, than the much-simplified experimental contexts that have served as intuitions for earlier work in the field; we have in mind here in particular the experiments discussed in (Ford and Olson, 1975), (Sonnenschein, 1985) and (Pechmann, 1989). In the belief that the data provides a good test set for the generation of referring expressions, we are making the data set publicly available [5], so others may try to develop algorithms covering the data.

A second concern is that we have only explored the extent to which three specific algorithms are able to cover the human data. Many of the other algorithms in the literature take these as a base, and so are unlikely to deliver significantly different results. The major exceptions here may be (a) van Deemter's (2002) algorithm for sets; recall that we excluded from the human data used here 16 references that involved sets; and, as noted above, (b) Krahmer et al's (2003) graph-based approach to GRE, which may perform better than the Relational Algorithm on descriptions using relations. In future work, we intend to explore to what extent our findings extend to other algorithms.

In conclusion, we point to two directions where we believe further work is required.

First, as we noted early in this paper, it is clear that there can be many different ways of referring to the same entity. Existing algorithms are all deterministic and therefore produce exactly one 'best' description for each entity; but the human-produced data clearly shows that there are many equally valid ways of describing an entity. We need to find some way to account for this in our algorithms. Our intuition is that this is likely to be best cashed out in terms of different 'reference strategies' that different speakers adopt in different situations; we are reminded here of Carletta's (1992) distinction between risky and cautious strategies for describing objects in the Map Task domain. More experimentation is required in order to determine just what these strategies are: are they, for example, characterisable as things like 'Produce a referring expression that is as short as possible' (the intuition behind the Full Brevity Algorithm), 'Just say what comes to mind first and keep adding information until the description distinguishes the intended referent' (something like the Incremental Algorithm), or perhaps a strategy of minimising the cognitive effort for either the speaker or the hearer? Further psycholinguistic experiments and data analysis are required to determine the answers here.

Our second observation is that the particular results we have presented here are, ultimately, en-

[5] The data set is publicly available from http://www.ics.mq.edu.au/∼jviethen/drawers

tirely dependent upon the underlying representations we have used, and the decisions we have made in choosing how to represent the properties and relations in the domain. We believe it is important to draw attention to the fact that precisely how we choose to represent the domain has an impact on what the algorithms will do. If we are aiming for naturalism in our algorithms for referring expression generation, then ideally we would like our representations to mirror those used by humans; but, of course, we don't have direct access to what these are.

There is clearly scope for psychological experimentation, perhaps along the lines initially explored by (Rosch, 1978), to determine some constraints here. In parallel, we are considering further exploration into the variety of representations that can be used, particularly with regard to the question of which properties are considered to be 'primitive', and which are generated by some inference mechanism; this is a much neglected aspect of the referring expression generation task.

References

D. E. Appelt. 1981. *Planning Natural Language Utterances to Satisfy Multiple Goals*. Ph.D. thesis, Stanford University.

J. Bateman. 1999. Using aggregation for selecting content when generating referring expressions. In *Proceedings of the 37th Meeting of the ACL*, pages 127–134.

J. C. Carletta. 1992. *Risk-taking and Recovery in Task-oriented Dialogue*. Ph.D. thesis, University of Edinburgh.

R. Dale and N. Haddock. 1991. Generating referring expressions involving relations. In *Proceedings of the 5th Meeting of the EACL*, pages 161–166, Berlin, Germany.

R. Dale and E. Reiter. 1995. Computational interpretations of the Gricean maxims in the generation of referring expressions. *Cognitive Science*, 19(2):233–263.

R. Dale. 1989. Cooking up referring expressions. In *Proceedings of the 27th Meeting of the ACL*, pages 68–75.

W. Ford and D. Olson. 1975. The elaboration of the noun phrase in children's description of objects. *Journal of Experimental Child Psychology*, 19:371–382.

K. Funakoshi, S. Watanabe, N. Kuriyama, and T. Tokunaga. 2004. Generating referring expressions using perceptual groups. In *Proceedings of the 3rd INLG*, pages 51–60.

C. Gardent. 2002. Generating minimal definite descriptions. In *Proceedings of the 40th Meeting of the ACL*, pages 96–103.

A. Gatt. 2006. Structuring knowledge for reference generation: A clustering algorithm. In *Proceedings of the 11th Meeting of the EACL*.

H. P. Grice. 1975. Logic and conversation. In P. Cole and J. Morgan, editors, *Syntax and Semantics Volume 3: Speech Acts*, pages 43–58. Academic Press.

H. Horacek. 1997. An algorithm for generating referential descriptions with flexible interfaces. In *Proceedings of the 35th Meeting of the ACL*, pages 127–134.

E. Krahmer and M. Theune. 2002. Efficient context-sensitive generation of referring expressions. In K. van Deemter and R. Kibble, editors, *Information Sharing: Reference and Presupposition in Language Generation and Interpretation*, pages 223–264. CSLI.

E. Krahmer, S. van Erk, and A. Verleg. 2003. Graph-based generation of referring expressions. *Computational Linguistics*, 29(1):53–72.

D. D. McDonald. 1980. *Natural Language Generation as a Process of Decision-making Under Constraints*. Ph.D. thesis, Massachusetts Institute of Technology.

T. Pechmann. 1989. Incremental speech production and referential overspecification. *Linguistics*, 27:89–110.

E. Reiter and R. Dale. 1992. A fast algorithm for the generation of referring expressions. In *Proceedings of the 14th Meeting of the ACL*, pages 232–238.

E. Rosch. 1978. Principles of categorization. In *Cognition and Categorization*, pages 27–48. Lawrence Erlbaum, Hillsdale, NJ.

S. Sonnenschein. 1985. The development of referential communication skills: Some situations in which speakers give redundant messages. *Journal of Psycholinguistic Research*, 14:489–508.

M. Stone. 2000. On identifying sets. In *Proceedings of the 1st INLG*, pages 116–123.

K. van Deemter and M. M. Halldórsson. 2001. Logical form equivalence: The case referring expressions generation. In *Proceedings of the 8th ENLG*.

K. van Deemter. 2002. Generating referring expressions: Boolean extensions of the incremental algorithm. *Computational Linguistics*, 28(1):37–52.

I. van der Sluis and E. Krahmer. 2004. Evaluating multimodal NLG using production experiments. In *Proceedings of the 4th LREC*, pages 209–212, 26-28 May.

T. Winograd. 1972. *Understanding Natural Language*. Academic Press.