# A Meta-Algorithm for the Generation of Referring Expressions

**Emiel Krahmer, Sebastiaan van Erk** and **André Verleg**
TU/e, Eindhoven University of Technology,
The Netherlands
email: E.J.Krahmer@tue.nl

## Abstract

This paper describes a new approach to the generation of referring expressions. We propose to formalize a *scene* as a labeled directed graph and describe *content selection* as a subgraph construction problem. Cost functions are used to guide the search process and to give preference to some solutions over others. The resulting graph algorithm can be seen as a meta-algorithm in the sense that defining cost functions in different ways allows us to mimic —and even improve— a number of well-known algorithms.

## 1 Introduction

The generation of referring expressions is one of the most common tasks in natural language generation, and has been addressed by many researchers in the past two decades (including Appelt 1985, Dale 1992, Reiter 1990, Dale & Haddock 1991, Dale & Reiter 1995, Horacek 1997, Stone & Webber 1998, Krahmer & Theune 1999 and van Deemter 2000). As a result, there are many different algorithms for the generation of referring expressions, each with its own objectives: some aim at producing the shortest possible description, others focus on efficiency or realistic output. The degree of detail in which the various algorithms are described differs considerably, and as a result it is often difficult to compare the various proposals. In addition, most of the algorithms are primarily concerned with the generation of descriptions only using properties of the target object. Consequently, the problem of generating relational descriptions (i.e., descriptions which incorporate references to other objects to single out the target object) has not received the attention it deserves.

In this paper, we describe a general, graph-theoretic approach to the generation of referring expressions. We propose to formalize a *scene* (i.e., a domain of objects and their properties and relations) as a labeled directed graph and describe the *content selection* problem —which properties and relations to include in a description for an object?— as a subgraph construction problem. The graph perspective has three main advantages. The first one is that there are many attractive algorithms for dealing with graph structures. In this paper, we describe a *branch and bound* algorithm for finding the relevant subgraphs, where we use *cost functions* to guide the search process. Arguably, the proposed algorithm is a *meta-algorithm*, in the sense that by defining the cost function in different ways, we can mimic various well-known algorithms for the generation of referring expressions. A second advantage of the graph-theoretical framework is that it does not run into problems with relational descriptions, due to the fact that properties and relations are formalized in the same way, namely as edges in a graph. The third advantage is that the combined usage of graphs and cost-functions paves the way for a natural integration of traditional rule-based approaches to generation with more recent statistical approaches (e.g., Langkilde & Knight 1998, Malouf 2000) in a single algorithm.

The outline of this paper is as follows. In section 2, we describe how scenes can be described as labeled directed graphs and show how content selection can be formalized as a subgraph construction problem. Section 3 contains a sketch of the branch and bound algorithm, which is illustrated with a worked example. In section 4 it is argued that by defining cost functions in different ways, we can mimic various well-known algorithms for the generation of referring expressions. We end with some concluding remarks in section 5.

## 2 Graphs

Consider the following scene:

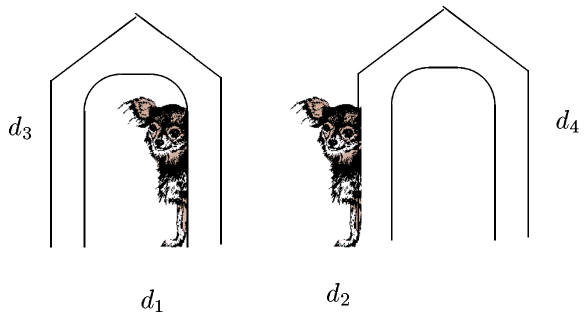

Figure 1: An example scene

In this scene, as in any other scene, we see a finite set of entities $E$ with properties $P$ and relations $R$. In this particular scene, the set $E = \{d_1, d_2, d_3, d_4\}$ is the set of entities, $P = \{$ dog, chihuahua, doghouse, small, large, white, brown $\}$ is the set of properties and $R = \{$ next_to, left_of, right_of, contain, in $\}$ is the set of relations. A scene can be represented in various ways. One common representation is to build a database, listing the properties of each element of $E$:

| | | | |
|---|---|---|---|
| $d_1$: | dog ($d_1$), | brown ($d_1$), ..., | in ($d_1$, $d_3$) |
| ⋮ | ⋮ | ⋮ | ⋮ |
| $d_4$: | doghouse ($d_4$), | white ($d_4$), ..., | right_of ($d_4$, $d_3$) |

Here we take a different approach and represent a scene as a *labeled directed graph.* Let $L = P \cup R$ be the set of labels (with $P$ and $R$ disjoint, i.e., $P \cap R = \emptyset$). Then, a labeled directed graph $G = \langle V, A \rangle$, where $V \subseteq E$ is the set of vertices (or nodes) and $A \subseteq V \times L \times V$ is the set of labeled directed arcs (or edges). The scene given in Figure 1 can be represented by the graph in Figure 2. Keep in mind that the $d$ labels are only added to ease reference to nodes. Notice also that properties (such as being a dog) are always modelled as loops, i.e., edges which start and end in the same node, while relations may (but need not) have different start and end nodes.

Now the content determination problem for referring expressions can be formulated as a graph construction task. In order to decide which information to include in a referring expression for an object $d \in V$, we construct a *connected* directed labeled graph over the set of labels $L$ and an arbitrary set of nodes, but including $d$. This graph can be understood as the "meaning representation" from which a referring expression can be generated by a linguistic realizer. Informally, we say that a graph *refers* to a given entity iff the graph can be "placed over" the scene graph in such a way that the node being referred to is "placed over" the given entity and each edge can be "placed over" an edge labeled with the same label. Furthermore, a graph is *distinguishing* iff it refers to exactly *one* node in the scene graph.

Consider the three graphs in Figure 3. Here and elsewhere circled nodes stand for the intended referent. Graph (i) refers to all nodes of the graph in Figure 2 (every object in the scene is next to some other object), graph (ii) can refer to both $d_1$ and $d_2$, and graph (iii) is distinguishing in that it can only refer to $d_1$. Notice that the three graphs might be realized as *something next to something else, a chihuahua* and *the dog in the doghouse* respectively. In this paper, we will concentrate on the generation of distinguishing graphs.

Formally, the notion that a graph $H = \langle V_H, A_H \rangle$ can be "placed over" another graph $G = \langle V_G, A_G \rangle$ corresponds to the notion of a *subgraph isomorphism.* $H$ can be "placed over" $G$ iff there exists a subgraph $G' = \langle V_{G'}, A_{G'} \rangle$ of $G$ such that $H$ is isomorphic to $G'$. $H$ is isomorphic to $G'$ iff there exists a bijection $\pi : V_H \to V_{G'}$ such that for all nodes $v, w \in V_H$ and all $l \in L$

$$(v, l, w) \in A_H \Leftrightarrow (\pi.v, l, \pi.w) \in A_{G'}$$

In words: the bijective function $\pi$ maps all the nodes in $H$ to corresponding nodes in $G'$, in such a way that any edge with label $l$ between nodes $v$

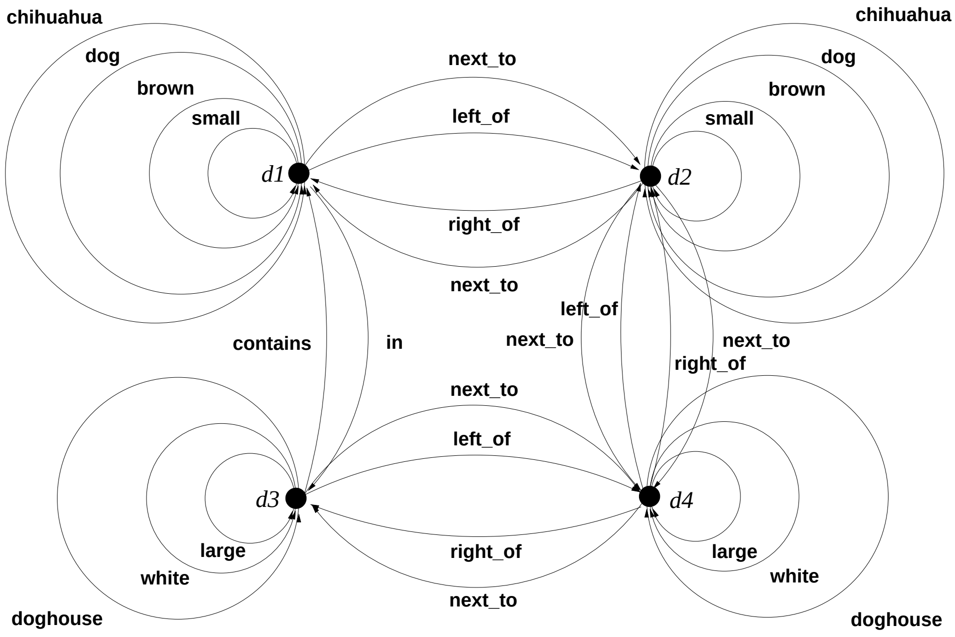

Figure 2: A graph representation of Figure 1.

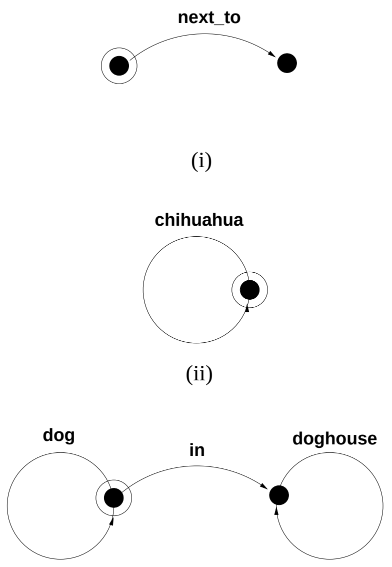

Figure 3: Some graphs for referring expressions, with circles around the intended referent.

and $w$ in $H$ is matched by an edge with the same label between the $G'$ counterparts of $v$ and $w$, i.e., $\pi.v$ and $\pi.w$ respectively. When $H$ is isomorphic to some subgraph of $G$ by an isomorphism $\pi$, we write $H \sqsubseteq_\pi G$.

Given a graph $H$ and a node $v$ in $H$, and a graph $G$ and a node $w$ in $G$, we define that the pair $(v, H)$ *refers* to the pair $(w, G)$ iff $H$ is connected and $H \sqsubseteq_\pi G$ and $\pi.v = w$. Furthermore, $(v, H)$ *uniquely refers* to $(w, G)$ (i.e., $(v, H)$ is *distinguishing*) iff $(v, H)$ refers to $(w, G)$ and there is no node $w'$ in $G$ different from $w$ such that $(v, H)$ refers to $(w', G)$. The problem considered in this paper can now be formalized as follows: given a graph $G$ and a node $w$ in $G$, find a pair $(v, H)$ such that $(v, H)$ uniquely refers to $(w, G)$.

Consider, for instance, the task of finding a pair $(v, H)$ which uniquely refers to the node labeled $d_1$ in Figure 2. It is easily seen that there are a *number* of such pairs, three of which are depicted in Figure 4. We would like to have a mechanism which allows us to give certain solutions preference over other solutions. For this purpose we shall use *cost-functions*. In general, a cost function $C$ is a function which assigns to each subgraph of a scene graph a positive number. As we shall see, by defining cost functions in different ways, we can mimic various algorithms for the generation of referring expressions known from the literature.

**A note on problem complexity** The basic decision problem for subgraph isomorphism (i.e., testing whether a graph $H$ is isomorphic to a subgraph of $G$) is known to be NP complete (see e.g., Garey & Johnson 1979). Here we are interested in *connected* $H$, but unfortunately that

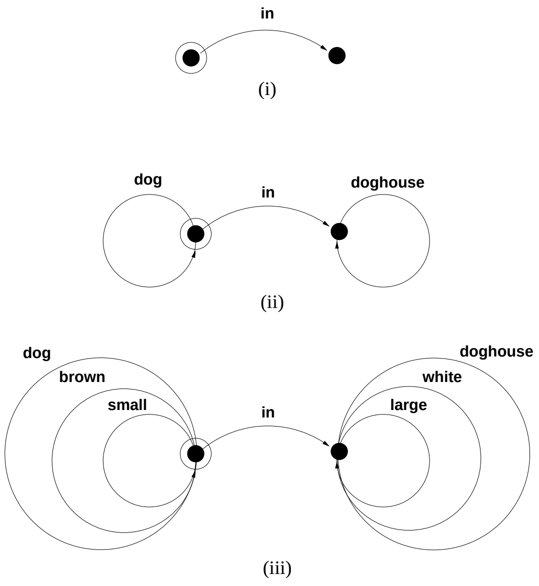

Figure 4: Three distinguishing node-graph pairs referring to $d_1$ in Figure 2.

restriction does not reduce the theoretical complexity. However, as soon as we define an upper bound $K$ on the number of edges in a distinguishing graph, the problem loses its intractability and becomes solvable in polynomial $\mathcal{O}(n^K)$ time. Such a restriction is rather harmless for our current purposes, as it would only prohibit the generation of distinguishing descriptions with more than $K$ properties, for an arbitrary large $K$. In general, there are various classes of graphs for which the subgraph isomorphism problem can be solved much more efficiently, without postulating upper bounds. For instance, if $G$ and $H$ are *planar graphs* the problem can be solved in time linear in the number of nodes of $G$ (Eppstein 1999). Basically, a planar graph is one which can be drawn on a plane in such a way that there are no crossing edges (thus, for instance, the graph in Figure 2 is planar). It is worth investigating to what extent planar graphs suffice for the generation of referring expressions.

## 3 Outline of the algorithm

In this section we give a high-level sketch of the algorithm. The algorithm (called **makeReferringExpression**) consists of two main components, a subgraph construction algorithm (called **findGraph**) and a subgraph isomorphism testing algorithm (called **matchGraphs**). We assume that a scene graph $G = \langle V, A\rangle$ is given. The algorithm systematically tries all relevant subgraphs $H$ of the scene graph by starting with the subgraph containing only the node $v$ (the target object) and expanding it recursively by trying to add edges from $G$ which are adjacent to the subgraph $H$ constructed so far. In this way we know that the results will be a *connected* subgraph. We refer to this set of adjacent edges as the $H$ neighbors in $G$ (notation: $G$.*neighbors*($H$)). The algorithm returns the cheapest distinguishing subgraph $H$ which refers to $v$, if such a distinguishing graph exists, otherwise it returns the empty graph $\bot = \langle\emptyset, \emptyset\rangle$.

### 3.1 Cost functions

We use cost functions to guide the search process and to give preference to some solutions over others. If $H = \langle V_H, A_H\rangle$ is a subgraph of $G$, then the costs of $H$, notation $C(H)$, are given by summing over the costs associated with the nodes and edges of H. Formally:

$$C(H) = \sum_{v \in V_H} C(v) + \sum_{e \in A_H} C(e)$$

We require the cost function to be *monotonic*. That is, adding an edge to a (non-empty) graph can never result in a *cheaper* graph. Formally:[1]

$$\forall G' \subseteq G : \forall e \in G.\mathit{edges}: \\ G'.\mathit{cost} \leq (G' + e).\mathit{cost}$$

This assumption helps reducing the search space substantially, since extensions of subgraphs with a cost greater than the best subgraph found so far can safely be ignored. The costs of the empty, undefined graph are infinite, i.e. $C(\bot) = \infty$.

### 3.2 Worked example

We now illustrate the algorithm with an example. Suppose the scene graph $G$ is as given in Figure 2, and that we want to generate a referring expression for object $d_1$ in this graph. Let us assume for the sake of illustration that the cost function is defined in such a way that adding a node or an edge always costs 1 point. Thus: for each $v \in V_G$ and for each $e \in A_H$: $C(v) = C(e) = 1$.

[1] Here and elsewhere, we use the following notation. Let $G = \langle V, A\rangle$ be a graph and $e$ an edge, then $G + e$ is the graph $\langle V \cup \{e.\mathit{node1}, e.\mathit{node2}\}, A \cup \{e\}\rangle$.

```
makeReferringExpression($v$) {
    bestGraph := $\perp$;
    $H$ := $\langle\{v\}, \emptyset\rangle$;
    return findGraph($v$, bestGraph, $H$);
}

findGraph($v$, bestGraph, $H$) {
    if (bestGraph.cost $\leq$ $H$.cost) then return bestGraph fi;
    distractors := $\{ n \mid n \in G.nodes \wedge \text{matchGraphs}(v, H, n, G) \wedge n \neq v\}$;
    if (distractors = $\emptyset$) then return $H$ fi;
    for each edge $\in$ $G$.neighbors($H$) do
        $I$ := findGraph($v$, bestGraph, $H + e$);
        if $I$.cost $\leq$ bestGraph.cost then bestGraph := $I$ fi;
    rof;
    return bestGraph;
}
```

Figure 5: Sketch of the main function (**makeReferringExpression**) and the subgraph construction function (**findGraph**).

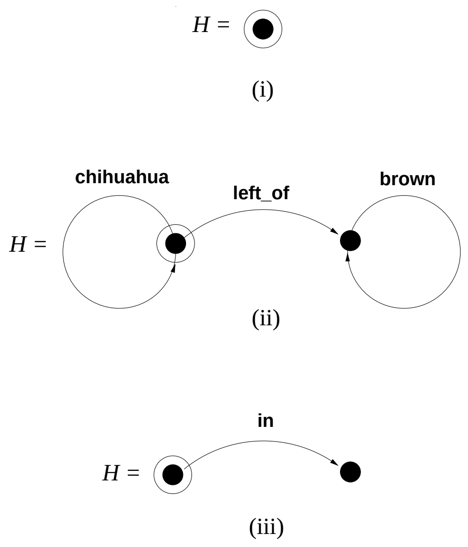

Figure 6: Three values for $H$ in the generation process for $d_1$.

(In the next section we describe a number of more interesting cost functions and discuss the impact these have on the output of the algorithm.) We call the function makeReferringExpression (given in Figure 5) with $d_1$ as parameter. In this function the variable *bestGraph* (for the best solution found so far) is initialized as the empty graph and the variable $H$ (for the distinguishing subgraph under construction) is initialized as the graph containing only node $d_1$ ((i) in Figure 6). Then the function findGraph (see also Figure 5) is called, with parameters $d_1$, *bestGraph* and $H$. In this function, first it is checked whether the costs of $H$ (the graph under construction) are higher than the costs of the *bestGraph* found so far. If that is the case, it is not worth extending $H$ since, due to the monotonicity constraint, it will never end up being cheaper than the current *bestGraph.* The initial value of *bestGraph* is the empty, undefined graph, and since its costs are astronomically high, we continue. Then the set of *distractors* (the objects from which the intended referent should be distinguished, Dale & Reiter 1995) is calculated. In terms of the graph perspective this is the set of nodes in the scene graph $G$ (other then the target node $v$) to which the graph $H$ refers. It is easily seen that the initial value of $H$, i.e., (i) in Figure 6, refers to every node in $G$. Hence, as one would expect, the initial set of distractors is $G.nodes - \{d_1\}$. Next we check whether the current set of distractors is empty. If so, we have managed to find a distinguishing graph, which is subsequently stored in the variable *bestGraph.* In this first iteration, this is obviously not the case and we continue, recursively trying to extend $H$ by adding adjacent (neighboring) edges until either a distinguishing graph has been constructed (all distract-

```
matchGraphs($v$, $H$, $w$, $G$) {
    if $H$.edges($v$, $v$) $\not\subseteq$ $G$.edges($w$, $w$) then return false fi;
    matching := $\{\pi.v = w\}$ ;
    $Y$ := $H$.neighbors($v$);
    return matchHelper(matching, $Y$, $H$);
}

matchHelper(matching, $Y$, $H$) {
    if | matching | $= |H|$ then return true fi;
    if $Y = \emptyset$ then return false fi;
    choose a fresh, unmatched $y$ from $Y$;
    $Z := \{z \in G \mid y$ might be matched to $z\}$;
    for each $z \in Z$ do
        if $z$ is a valid extension of the mapping
        then if matchHelper(matching $\cup\{\pi.y = z\}$, $Y$, $H$) then return true fi;
        fi;
    rof;
    return false;
}
```

Figure 7: Sketch of the function testing for subgraph isomorphism (**matchGraphs**).

ors are ruled out) or the costs of $H$ exceed the costs of the *bestGraph* found so far. While *bestGraph* is still the empty set (i.e., no distinguishing graph has been found yet), the algorithm continues until $H$ is a distinguishing graph. Which is the first distinguishing graph to be found (if one or more exist) depends on the order in which the adjacent edges are tried. Suppose for the sake of argument that the first distinguishing graph to be found is (ii) in Figure 6. This graph is returned and stored in *bestGraph*. The costs associated with this graph are 5 points (two nodes and three edges). At this stage in the generation process only graphs with *lower* costs are worth investigating, which yields a drastic reduction of the search space. In fact, there are only a few distinguishing graphs which cost less. After a number of iterations the algorithm will find the cheapest solution (given this particular, simple definition of the cost function), which is (iii) in Figure 6.

## 3.3 Subgraph Isomorphism testing

Figure 7 contains a sketch of the part of the algorithm which tests for subgraph isomorphism, matchGraphs. This function is called each time the distractor set is calculated. It tests whether the pair $(v, H)$ can refer to $(w, G)$, or put differently, it checks whether there exists an isomorphism $\pi$ such that $H \sqsubseteq_\pi G$ with $\pi.v = w$. The function matchGraphs first determines whether the looping edges starting from node $v$ (i.e., the properties of $v$) match those of $w$. If not (e.g., $v$ is a dog and $w$ is a doghouse), we can immediately discard the matching. Otherwise we start with the matching $\pi.v = w$, and expand it recursively. Each recursion step a fresh and as yet unmatched node $y$ from $H$ is selected which is adjacent to one of the nodes in the current matching. For each $y$ we calculate the set $Z$ of possible nodes in $G$ to which $y$ can be matched. This set consist of all the nodes in $G$ which have the same looping edges as $y$ and the same edges to and from other nodes in the domain of the current matching function $\pi$:

$$\begin{aligned} Z := \{z \mid\ & z \in G.nodes \wedge \\ & H.edges(y, y) \subseteq G.edges(z, z) \wedge \\ & \forall h \in H.neighbors(y) \cap Dom(\pi) : \\ & (H.edges(y, h) \subseteq G.edges(z, \pi.h) \wedge \\ & \ H.edges(h, y) \subseteq G.edges(\pi.h, z)) \\ \} \end{aligned}$$

The matching can now be extended with $\pi.y = z$, for $z \in Z$. The algorithm then branches over all these possibilities. Once a mapping $\pi$ has been found which has exactly as much elements as $H$ has nodes, we have found a subgraph isomorphism. If there are still unmatched nodes in $H$ or if all possible extensions with a node $y$ have been checked and no matching could be found, the test for subgraph isomorphism has failed.

### 3.4 A note on the implementation

The basic algorithm outlined in Figures 5 and 7 has been implemented in Java. Various optimizations increase the efficiency of the algorithm, as certain calculations need not be repeated each iteration (e.g., the set $G$.*neighbors*($H$)). In addition, the user has the possibility of specifying the cost function in a way which he or she sees fit.

## 4 Search methods and cost functions

Arguably, the algorithm outlined above is a *meta*-algorithm, since by formulating the cost function in certain ways we can simulate various algorithms known from the generation literature.

### 4.1 Full (relational) Brevity Algorithm

The algorithm described in the previous section can be seen as a generalization of Dale's (1992) *Full Brevity* algorithm, in the sense that there is a guarantee that the algorithm will output the shortest possible description, if one exists. It is also an *extension* of the Full Brevity algorithm, since it allows for relational descriptions, as does the Dale & Haddock (1991) algorithm. The latter algorithm has a problem with infinite recursions; in principle their algorithm could output descriptions like "the dog in the doghouse which contains a dog which is in a doghouse which …etc." Dale & Haddock propose to solve this problem by stipulating that a property or relation may only be included once. In the graph-based model described above the possibility of such infinite recursions does not arise, since a particular edge is either present in a graph or not.[2]

[2] Notice incidentally that Dale's (1992) *Greedy Heuristic* algorithm can also be cast in the graph framework, by sorting edges on their descriptive power (measured as a count of the number of occurrences of this particular edge in the scene graph). The algorithm then adds the most discriminating edge first (or the cheapest, if there are various equally distinguishing edges) and repeats this process until a distinguishing graph is found.

### 4.2 Incremental Algorithm

Dale & Reiter's (1995) Incremental Algorithm, generally considered the state of the art in this field, has the following characteristic properties. (1) It defines a list of *preferred attributes*, listing the attributes which human speakers prefer for a certain domain. For example, when discussing domestic animals, speakers usually first describe the "type" of animal (dog, cat), before *absolute* properties such as "color" are used. If that still is not sufficient to produce a distinguishing description, *relative* properties such as "size" can be included. Thus, the list of preferred attributes for this particular domain could be ⟨ type, color, size ⟩. The Incremental Algorithm now simply iterates through this list, adding a property if it rules out any distractors not previously ruled out. (2) The algorithm always includes the "type" attribute, even if it is not distinguishing. And (3) the algorithm allows subsumption hierarchies on certain attributes (most notably for the "type" attribute) stating things like a fox terrier is a dog, and a dog is an animal. In such a hierarchy we can specify what the *basic level* value is (in this case it is dog). Dale & Reiter claim that there is a general preference for basic level values, and hence their algorithm includes the basic level value of an attribute, unless values subsumed by the basic level value rule out *more* distractors.

These properties can be incorporated in the graph framework in the following way. (1) The list of preferred attributes can easily be modelled using the cost function. All "type" edges should be cheaper than all other edges (in fact, they could be for free), and moreover, the edges corresponding to absolute properties should cost less than those corresponding to relative ones. This gives us exactly the effect of having preferred attributes. (2) It also implies that the "type" of an object is always included if it is in any way distinguishing. That by itself does not guarantee that type is *always* is included. The most principled and efficient way to achieve that would be to reformulate the findGraph algorithm in such a way that the "type" loop is always included. (Given such a minor modification, the algorithm described in the previous section would output (iii) from Figure 3 instead of (iii) from Figure 6 when applied to $d_1$.) Such a general modification might be undesirable from an empirical point of view however, since in various domains it is very common to not include type information, for instance when the domain contains only objects of the same type (see van der Sluis & Krahmer 2001). (3) The subsumption hierarchy can be modelled in the same way as preferred attributes are: for a given attribute, the basic level value should have the lowest costs and the values farthest away from the basic level value should have the highest costs. This implies that adding an edge labeled dog is cheaper than adding an edge labeled chihuahua, unless more (or more expensive) edges are needed to build a distinguishing graph including dog than are required for the graph including chihuahua. Assuming that the scene representation is well-defined, the algorithm never outputs a graph which contains *both* dog *and* chihuahua, since there will always be a cheaper distinguishing graph omitting one of the two edges.

So, we can recast the Incremental Algorithm quite easily in terms of graphs. Note that the original Incremental Algorithm only operates on properties, looped edges in graph terminology. It is worth stressing that when all edges in the scene graph are of the looping variety, testing for subgraph isomorphism becomes trivial and we regain polynomial complexity. However, the above graph-theoretical formalization of the Incremental Algorithm does not fully exploit the possibilities offered by the graph framework and the use of cost functions. First, from the graph-theoretical perspective the generation of relational descriptions poses no problems whatsoever, while the incremental generation of relational descriptions is by no means trivial (see e.g., Theune 2000, Krahmer & Theune 1999). In fact, while it could be argued to some extent that incremental selection of properties is psychologically plausible, this somehow seems less plausible for incremental generation of relational extensions.[3] Notice that the use of a cost function to simulate subsumption hierarchies for properties carries over directly to relations; for instance, the costs of adding a edge labeled next_to should be less than those of adding one labeled left_of or right_of. Hence, next_to will be preferred, unless using left_of or right_of has more discriminative power. Another advantage of the way the graph-based algorithm models the list of preferred attributes is that more fine-grained distinctions can be made than can be done in the Incremental Algorithm. In particular, we are not forced to say that values of the attribute "type" are *always* preferred over values of the attribute "color". Instead we have the freedom to assign edges labeled with a common type value (e.g., dog) a lower cost than edges labeled with uncommon colors (such as Vandyke-brown), while at the same time edges labeled with obscure type values, such as polish owczarek nizinny sheepdog, can be given a *higher* cost than edges labeled with common colors such as brown.

[3]As Dale & Reiter (1995:248) point out, redundant properties are not uncommon. That is: in certain situations people may describe an object as "the white bird" even though the simpler "the bird" would have been sufficient (cf. Pechmann 1989, see also Krahmer & Theune 1999 for discussion). However, a similar argument seems somewhat far-fetched when applied to relations. It is unlikely that someone would describe an object as "the dog next to the tree in front of the garage" in a situation where "the dog in front of the garage" would suffice.

## 4.3 Stochastic cost functions

One of the important open questions in natural language generation is how the common, rule-based approaches to generation can be combined with recent insights from statistical NLP (see e.g., Langkilde & Knight 1998, Malouf 2000 for partial answers). Indeed, when looking at the Incremental Algorithm, for instance, it is not directly obvious how statistical information can be integrated in the algorithm. Arguably, this is different when we have cost functions. One can easily imagine deriving a stochastic cost function from a sufficiently large corpus and using it in the graph-theoretical framework (the result looks like but is not quite a Markov Model). As a first approximation, we could define the costs of adding an edge $C(e)$ in terms of the probability $P(e)$ that $e$ occurs in a distinguishing description (estimated by counting occurrences):

$$C(e) = -\log_2(P(e))$$

Thus, properties which occur frequently are cheap, properties which are relatively rare are expensive. In this way, we would probably *derive* that dog is indeed less expensive than Vandyke brown and that brown is less expensive than polish owczarek nizinny sheepdog.

## 5 Concluding remarks

In this paper, we have presented a general graph-theoretical approach to content-determination for referring expressions. The basic algorithm has clear computational properties: it is NP complete, but there exist various modifications (a ban on non-looping edges, planar graphs, upper bound to the number of edges in a distinguishing graph) which make the algorithm polynomial. The algorithm is fully implemented. The graph perspective has a number of attractive properties. The generation of relational descriptions is straightforward; the problems which plague some other algorithms for the generation of relational descriptions do not arise. The use of cost functions allows us to model different search methods, each restricting the search space in its own way. By defining cost functions in different ways, we can model and extend various well-known algorithms from the literature such as the Full Brevity Algorithm and the Incremental Algorithm. In addition, the use of cost functions paves the way for integrating statistical information directly in the generation process.[4]

Various important ingredients of other generation algorithms can be captured in the algorithm proposed here as well. For instance, Horacek (1997) points out that an algorithm should not collect a set of properties which cannot be realized given the constraints of the grammar. This problem can be solved, following Horacek's suggestion, by slightly modifying the algorithm in such a way that for each potential edge it is immediately investigated whether it can expressed by the realizer. Van Deemter's (2000) proposal to generate (distributional) distinguishing plural descriptions (such as *the dogs*) can also be modelled quite easily. Van Deemter's algorithm takes as input a set of objects, which in our case, translates into a set of nodes from the scene graph. The algorithm should be reformulated in such a way that it tries to generate a subgraph which can refer to each of the nodes in the set, but not to any of the nodes in the scene graph outside this set. Krahmer & Theune (1999) present an extension of the Incremental Algorithm which takes context into account. They argue that an object which has been mentioned in the recent context is somehow salient, and hence can be referred to using fewer properties. This is modelled by assigning salience weights to objects (basically using a version of Centering Theory (Grosz et al. 1995) augmented with a recency effect), and by defining the set of distractors as the set of objects with a salience weight higher or equal than that of the target object. In terms of the graph-theoretical framework, one can easily imagine assigning salience weights to the nodes in the scene graph, and restricting the distractor set essentially as Krahmer & Theune do. In this way, distinguishing graphs for salient objects will generally be smaller than those of non-salient objects.

[4] A final advantage of the graph model certainly deserves further investigation is the following. We can look at a graph such as that in Figure 2 as a *Kripke model*. The advantage of this way of looking at it, is that we can use tools from modal logic to *reason* about these structures. For example, we can reformulate the problem of determining the content of a distinguishing description in terms of hybrid logic (see e.g., Blackburn 2000) as follows:

$$@_i\varphi \wedge \mathrm{A}j(i \neq j \rightarrow \neg @_j\varphi)$$

In words: when we want to refer to node $i$, we are looking for that distinguishing formula $\varphi$ which is true of ("at") $i$ but not of any $j$ different from $i$. One advantage of this perspective is that logical properties which are usually considered problematic from a generation perspective (such as *not* having a certain property), fit in very well with the logical perspective.

### Acknowledgements

Thanks are due to Alexander Koller, Kees van Deemter, Paul Piwek, Mariët Theune and two anonymous referees for discussions and comments on an earlier version of this paper.

## References

Appelt, D. (1985), Planning English Referring Expressions, *Artificial Intelligence* 26:1-33.

Blackburn, P. (2000), Representation, Reasoning, and Relational Structure: A Hybrid Logic Manifesto, *Logic Journal of the IGPL* 8(3):339-365.

Dale, R. (1992), *Generating Referring Expressions: Constructing Descriptions in a Domain of Objects and Processes*, MIT Press, Cambridge, Massachusetts.

Dale, R. & N. Haddock (1991), Generating Referring Expressions Involving Relations, *Proceedings of EACL*, Berlin, 161-166.

Dale, R. & E. Reiter (1995), Computational Interpretations of the Gricean Maxims in the Generation of Referring Expressions, *Cognitive Science* 18: 233-263.

van Deemter, K. (2000), Generating Vague Descriptions, *Proceedings INLG*, Mitzpe Ramon.

Eppstein, D. (1999), Subgraph Isomorphism in Planar Graphs and Related Problems, *J. Graph Algorithms and Applications* 3(3):1-27.

Garey, M. & D. Johnson (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman.

Grosz, B., A. Joshi & S. Weinstein (1995), Centering: A Framework for Modeling the Local Coherence of Discourse, *Computational Linguistics* 21(2):203-225.

Horacek, H. (1997), An Algorithm for Generating Referential Descriptions with Flexible Interfaces, *Proceedings of the 35th ACL/EACL*, Madrid, 206-213.

Krahmer, E. & M. Theune (1999), Efficient Generation of Descriptions in Context, *Proceedings of Workshop on Generation of Nominals*, R. Kibble and K. van Deemter (eds.), Utrecht, The Netherlands.

Langkilde, I. & K. Knight (1998), The Practical Value of $n$-Grams in Generation, *Proceedings INLG*, Niagara-on-the-lake, Ontario, 248-255.

Malouf, R., (2000), The Order of Prenominal Adjectives in Natural Language Generation, *Proceedings of the 38th ACL* , Hong Kong.

Pechmann, T. (1989), Incremental Speech Production and Referential Overspecification, *Linguistics* 27:98–110.

Reiter, E. (1990), The Computational Complexity of Avoiding Conversational Implicatures, *Proceedings of the 28th ACL* , 97-104.

van der Sluis, I. & E. Krahmer (2001), Generating Referring Expressions in a Multimodal Context: An Empirically Motivated Approach, *Proceedings CLIN*, W. Daelemans et al. (eds), Rodopi, Amsterdam/Atlanta.

Stone, M. & B. Webber (1998), Textual Economy Through Close Coupling of Syntax and Semantics, *Proceedings INLG*, Niagara-on-the-lake, Ontario, 178-187.

Theune, M. (2000), *From Data to Speech: Language Generation in Context*, Ph.D. dissertation, Eindhoven University of Technology.