Graph-Based Generation of Referring Expressions

Emiel Krahmer*
Tilburg University

Sebastiaan van Erk†
Eindhoven University of Technology

André Verleg‡
Eindhoven University of Technology

This article describes a new approach to the generation of referring expressions. We propose to formalize a scene (consisting of a set of objects with various properties and relations) as a labeled directed graph and describe content selection (which properties to include in a referring expression) as a subgraph construction problem. Cost functions are used to guide the search process and to give preference to some solutions over others. The current approach has four main advantages: (1) Graph structures have been studied extensively, and by moving to a graph perspective we get direct access to the many theories and algorithms for dealing with graphs; (2) many existing generation algorithms can be reformulated in terms of graphs, and this enhances comparison and integration of the various approaches; (3) the graph perspective allows us to solve a number of problems that have plagued earlier algorithms for the generation of referring expressions; and (4) the combined use of graphs and cost functions paves the way for an integration of rule-based generation techniques with more recent stochastic approaches.

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), Reiter (1990), Dale and Haddock (1991), Dale (1992), Dale and Reiter (1995), Horacek (1997), Stone and Webber (1998), Krahmer and Theune (1998, 2002), Bateman (1999), and van Deemter (2000, 2002). In this article, we present 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 four main advantages. (1) There are many attractive and well-understood algorithms for dealing with graph structures (see, e.g., Gibbons [1985], Cormen, Leiserson, and Rivest [1990], or Chartrand and Oellermann [1993]). In this article, we describe a straightforward **branch and bound** algorithm for finding the relevant subgraphs in which **cost functions** are used to guide the search process. (2) By defining different cost functions for the graph perspective, we can simulate (and improve) some of the well-known algorithms for the generation of referring

* Communication and Cognition/Computational Linguistics, Faculty of Arts, Tilburg University, Tilburg, The Netherlands. E-mail: E.J.Krahmer@uvt.nl.
† Tijgerstraat 2, NL-5645 CK, Eindhoven, The Netherlands. E-mail: sebster@sebster.com.
‡ Ranonkelstraat 67, NL-5644 LB, Eindhoven, The Netherlands. E-mail: andre@astygian.nl.

expressions mentioned above. This facilitates the formal comparison of these algorithms and makes it easier to transfer results from one algorithm to another. (3) The graph perspective provides a clean solution for some problems that have plagued earlier algorithms. For instance, the generation of **relational** expressions (i.e., referring expressions that include references to other objects) is enhanced by the fact that both properties and relations are formalized in the same way, namely, as edges in a graph. (4) The combined use of graphs and cost functions paves the way for a natural integration of traditional rule-based approaches to generating referring expressions and more recent statistical approaches, such as Langkilde and Knight (1998) and Malouf (2000), in a single algorithm.

The outline of this article is as follows. In Section 2 the content selection problem for generating referring expressions is explained, and some well-known solutions to the problem are discussed. In Section 3, we describe how scenes can be modeled as labeled directed graphs and show how content selection can be formalized as a subgraph construction problem. Section 4 contains a sketch of the basic generation algorithm, which is illustrated with a worked example. In Section 5 various ways to formalize cost functions are discussed and compared. We end with some concluding remarks and a discussion of future research directions in Section 6.

2. Generating Referring Expressions

There are many different algorithms for the generation of referring expressions, each with its own objectives: Some aim at producing the shortest possible description (e.g., Dale's [1992] full brevity algorithm), others focus on psychological realism (e.g., Dale and Reiter's [1995] incremental algorithm) or realistic output (e.g., Horacek 1997). The degree of detail in which the various algorithms are described differs considerably. Some algorithms are fully formalized and come with explicit characterizations of their complexity (e.g., Dale and Reiter 1995; van Deemter 2000); others are more conceptual and concentrate on exploring new directions (e.g., Stone and Webber 1998). Despite such differences, most algorithms deal with the same problem definition. They take as input a single object v (the **target object**) for which a referring expression is to be generated and a set of objects (the **distractors**) from which the target object needs to be distinguished (we use the terminology from Dale and Reiter [1995]). The task of the algorithm is to determine which set of properties is needed to single out the target object v from the distractors. This is known as the **content determination** problem for referring expressions. On the basis of this set of properties a **distinguishing description** for v can be generated. Most algorithms do not address the **surface realization** problem (how the selected properties should be realized in natural language) in much detail; it is usually assumed that once the content for a referring expression has been determined, a standard realizer such as KPML (Bateman 1997) or SURGE (Elhaded and Robin 1997) can convert the meaning representation to natural language.

Consider the example scene in Figure 1. In this scene, as in any other scene, we see a finite domain of entities D with properties P. In this particular scene, $D = \{d_1, d_2, d_3, d_4\}$ is the set of entities and $P = \{$ dog, cat, brown, black+white, large, small $\}$ is the set of properties. A scene is usually represented as a database (or knowledge base) listing the properties of each element in D. Thus:

d_1:	dog (d_1)	small (d_1)	brown (d_1)
d_2:	dog (d_2)	large (d_2)	brown (d_2)
d_3:	dog (d_3)	large (d_3)	black+white (d_3)
d_4:	cat (d_4)	small (d_4)	brown (d_4)

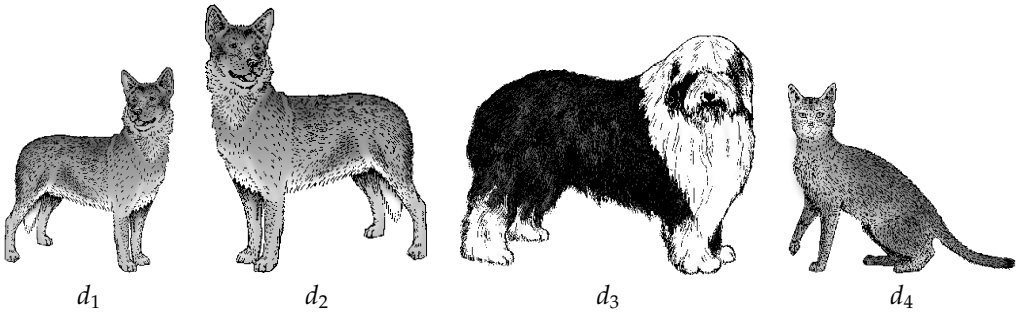

Figure 1
A simple example scene consisting of some domestic animals.

In what is probably the key reference on the topic, Dale and Reiter (1995) describe and discuss a number of algorithms for the generation of referring expressions. One of these is the **full brevity algorithm** (originally due to Dale 1992). This algorithm first tries to generate a distinguishing description for the target object v using one *single* property. If this fails, it considers all possible combinations of *two* properties to see if any of these suffices for the generation of a distinguishing description, and so on. It is readily seen that this algorithm will output the shortest possible description, if one exists. Suppose the full brevity algorithm is used to generate a description for d_1 in Figure 1. There is no *single* property that distinguishes the target object d_1 from the distractors $\{d_2, d_3, d_4\}$. But when considering all pairs of properties the algorithm will find that one such pair rules out all distractors, namely, small and dog; "the small dog" is a successful and minimal distinguishing description for d_1.

Dale and Reiter point out that the full brevity algorithm is both computationally infeasible (NP hard) and psychologically unrealistic. They offer the **incremental algorithm** as an alternative. The incremental algorithm considers properties for selection in a predetermined order, based on the idea that human speakers and listeners prefer certain *kinds* of properties (or **attributes**) when describing objects from a given domain. For instance, when discussing domestic animals, it seems likely that a human speaker would first describe an animal by its *type* (is it a dog? is it a cat?). If that does not suffice, first **absolute** attributes like *color* are tried, followed by **relative** ones such as *size*. In sum: The list of preferred attributes for our example domain would be ⟨ type, color, size ⟩. Essentially, the incremental algorithm iterates through this list, and for each property it encounters, it determines whether adding this property to the properties selected so far would rule out any of the remaining distractors. If so, it is included in the list of selected properties. There is one exception to this general strategy: Type information is *always* included, even if it rules out no distractors. The algorithm stops when all distractors are ruled out (**success**) or when the end of the list of preferred attributes is reached (**failure**).

Suppose we apply the incremental algorithm to d_1 from Figure 1 with ⟨ type, color, size ⟩ as preferred attributes. The type of d_1 listed in the database is dog. This property is selected (since type information is always selected). It rules out d_4 (which is a cat). Next we consider the color of d_1; the animal is brown. This property rules out d_3 (which is a *black and white* dog) and is selected. Finally, we consider the size of our target object, which is small. This properly rules out the remaining distractor d_2 (which is a *large* brown dog) and hence is included as well. At this point, all distractors are ruled out (success!), and the set of selected properties is {dog, brown, small}, which a linguistic realizer might express as "the small brown dog." This is a successful distinguishing

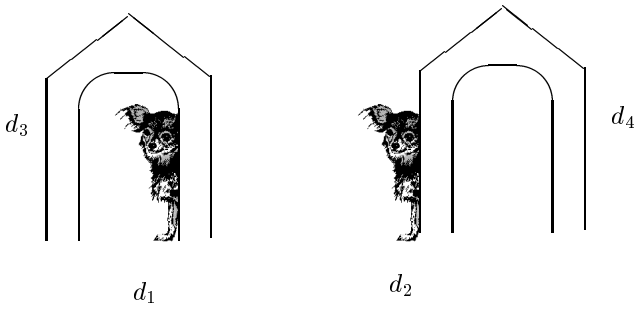

Figure 2
Another scene: Two dogs and two doghouses (from Krahmer and Theune [2002]).

description but *not* a minimal one: The property brown is, strictly speaking, made redundant by the later inclusion of the property small. Since there is no backtracking in the incremental algorithm however, *every* selected property is realized (hence "incremental"). This aspect is largely responsible for the computational efficiency of the algorithm (it has a polynomial complexity), but Dale and Reiter (1995, page 248) also claim that it is "psychologically realistic." They point out that sometimes people may describe an object as "the white bird" even though the simpler "the bird" would have been sufficient (cf. Pechmann [1989]; see, however, Krahmer and Theune [2002] for discussion).

Even though there are various useful and interesting algorithms for the generation of referring expressions, a number of open questions remain. Recently there has been an increased interest in statistical approaches to natural language generation. For example, Malouf (2000) has shown that large corpora can be used to determine the order of realization of sequences of prenominal adjectives. It is unclear how such statistical work on generation can be combined with older, rule-based work such as the algorithms just discussed. In addition, many algorithms still have difficulties with the generation of relational descriptions (descriptions that include references to other objects to single out the target object from its distractors). To illustrate the problem, consider the scene depicted in Figure 2. In this scene we again see a finite domain of entities D with certain properties P. Here, $D = \{d_1, d_2, d_3, d_4\}$ is the set of entities, and $P = \{$ dog, doghouse, small, large, brown, white $\}$ is the set of properties. Clearly no algorithm can generate a distinguishing description referring to d_1 on this basis. Intuitively, d_1 can be distinguished from d_2 only using its relation to the doghouse d_3. To facilitate this we extend the scene description with a set of relations $R = \{$ left_of, right_of, contain, in $\}$.

A few algorithms have been developed that address the issue of relational descriptions. The earliest is from Dale and Haddock (1992), who offer an extension of the full brevity algorithm. The Dale and Haddock algorithm has a problem with **infinite recursions**; it may produce descriptions like "the dog in the doghouse that contains a dog that is inside a doghouse. . . ." Dale and Haddock, somewhat ad hoc, solve this problem by stipulating that a property or relation may be used only once. Krahmer and Theune (2002) (see also Theune [2000]) describe an extension of the incremental algorithm that allows for relational descriptions. Their extension suffers from what may be called the problem of **forced incrementality**: When a first relation fails to rule out all remaining distractors, additional relations will be tried incrementally. Although it could be argued that incremental selection of properties is psychologically plausible, it seems less plausible for relations. It is unlikely that someone would describe an

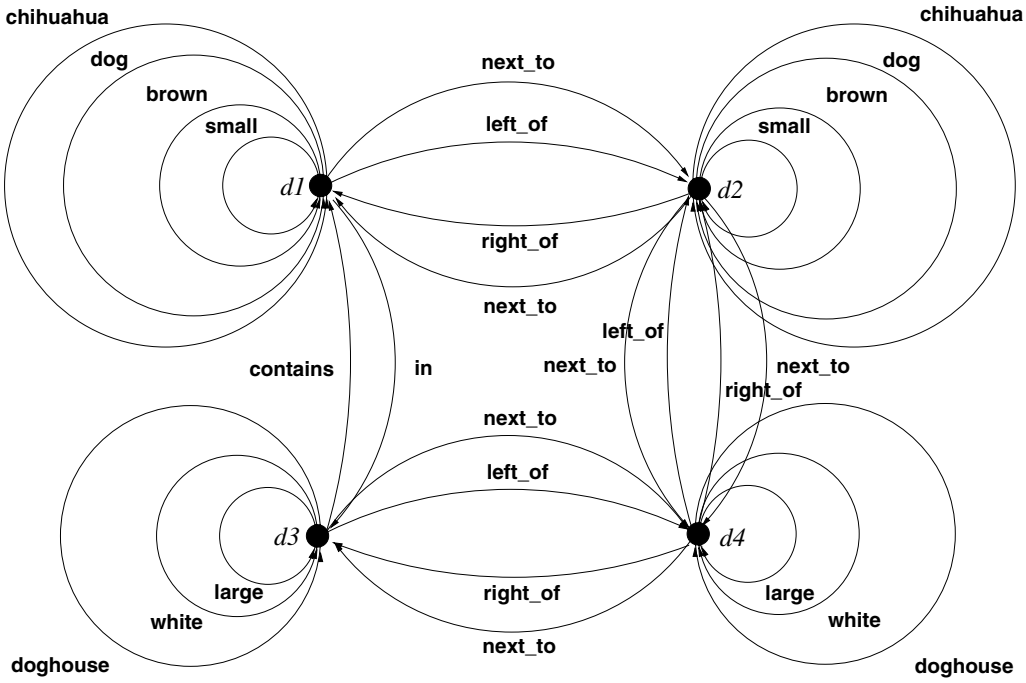

Figure 3
A graph representation of the scene in Figure 2.

object as "the dog next to the tree in front of the garage" in a situation in which "the dog in front of the garage" would suffice. As we shall argue, the graph perspective provides a clean solution for these problems.

3. Generating Referring Expressions Using Graphs

In the previous section we saw that a scene can be described in terms of a domain of entities D with properties P and relations R. Such a scene can be represented as a **labeled directed graph** (see, e.g., Wilson [1996] for a gentle introduction or Berge [1985] for a more specialized one). Let $L = P \cup R$ be the set of labels with P and R disjoint (i.e., $P \cap R = \emptyset$). Then $G = \langle V_G, E_G \rangle$ is a labeled directed graph, where $V_G \subseteq D$ is the set of **vertices** (or **nodes**) and $E_G \subseteq V_G \times L \times V_G$ is the set of **labeled directed edges** (or **arcs**). Where this can be done without creating confusion, the graph subscript is omitted. Throughout this article we use the following notations. If $G = \langle V, E \rangle$ is a graph and $e = \langle v, l, w \rangle$ an edge (with $l \in L$), then the **extension** of G with e, denoted as $G + e$, is the graph $\langle V \cup \{v, w\}, E \cup \{e\} \rangle$. Moreover, with $E_G(v, w)$ we refer to the set of edges in E_G from v to w; that is, $E_G(v, w) = \{e \in E_G \mid e = \langle v, l, w \rangle$, for $l \in L\}$.

The scene given in Figure 2, for example, can now be represented by the graph in Figure 3. This graph models the respective spatial relations between the two chihuahuas, between the two doghouses, and between each dog and the nearest doghouse. For the sake of transparency we have not modeled the relations between the dogs and the distant doghouses (i.e., between d_1 and d_4 and between d_2 and d_3). (It is worth stressing that adding these edges would not result in different outcomes in the discussion below). Note that properties (such as dog) are always modeled as loops,

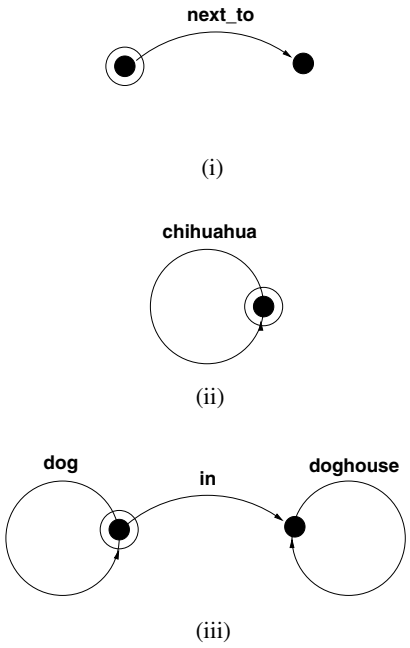

Figure 4
Some graphs for referring expressions, with circles around the intended referent.

that is, as edges that start and end in the same vertex. Relations may have different start and end vertices, but they do not have to (consider potentially reflexive relations such as shave). Finally, note that the graph sometimes contains properties of various levels of specificity (e.g., chihuahua and dog). This aspect of scene graphs will be further discussed in Section 5.

Now the content determination problem for referring expressions can be formulated as a graph construction problem. To decide which information to include in a referring expression for an object $v \in V$, we construct a **connected** directed labeled graph over the set of labels L and an arbitrary set of vertices, but including v. A graph is connected iff there is a **path** (a list of vertices in which each vertex has an edge from itself to the next vertex) between each pair of vertices. Informally, we say that a vertex ("the intended referent") from a graph H **refers** to a given entity in the scene graph G iff the graph H can be "placed over" the scene graph G in such a way that the vertex being referred to is "placed over" the vertex of the given entity in G and each edge from H with label l can be "placed over" an edge from G with the same label. Furthermore, a vertex-graph pair is **distinguishing** iff it refers to exactly *one* vertex in the scene graph.

Consider the three vertex-graph pairs in Figure 4, in which circled vertices stand for the intended referent. Graph (i) refers to all vertices of the graph in Figure 3 (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 refer only to d_1. Note that the graphs might be realized as *something next to something else*, *a chihuahua*, and *the dog in the doghouse*, respectively. Here we concentrate on the generation of distinguishing vertex-graph pairs.

Formally, the notion that a graph $H = \langle V_H, E_H \rangle$ can be "placed over" another graph $G = \langle V_G, E_G \rangle$ corresponds to the notion of a **subgraph isomorphism** (see, e.g.,

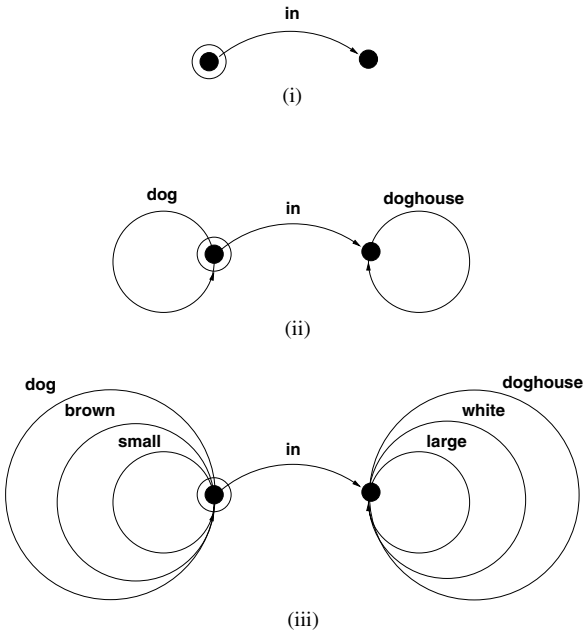

Figure 5
Three distinguishing vertex-graph pairs referring to d_1 in Figure 3.

Read and Corneil [1977] for an overview). H can be "placed over" G iff there exists a subgraph $G' = \langle V_{G'}, E_{G'} \rangle$ of G such that H is *isomorphic* to G'. H is isomorphic to G' iff there exists a bijection π: $V_H \rightarrow V_{G'}$ such that for all vertices $v, w \in V_H$ and all $l \in L$:

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

In words: The bijective function π maps all the vertices in H to corresponding vertices in G' in such a way that any edge with label l between vertices v and w in H is matched with 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 π, we write $H \sqsubseteq_\pi G$.

Given a graph H and a vertex v in H, and a graph G and a vertex 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 vertex w' in G different from w such that (v, H) refers to (w', G). The problem considered in this article can now be formalized as follows: Given a graph G and a vertex 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) that uniquely refers to the vertex labeled d_1 in Figure 3. It is easily seen that there are a *number* of such pairs, three of which are depicted in Figure 5. We would like to have a mechanism that allows us to give certain solutions to this kind of task preference over other solutions. For this purpose we shall use **cost functions**. In general, a cost function is a function that assigns to each subgraph of a scene graph a non-negative 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.

3.1 A Note on the 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 and Johnson [1979]). Here we are interested in *connected* H, but unfortunately that restriction does not reduce the theoretical complexity. Note that this characterization of the *worst-case* complexity holds for graphs in which all edges have the same label; in that case each edge from H can potentially be matched to any edge from G. The *best-case* complexity is given when each edge is uniquely labeled. In practice, the situation will most often be somewhere between these extremes. In general, we can say that the more diverse the labeling of edges in the graph of a particular scene is, the sooner a distinguishing vertex-graph pair will be found.

It is worth pointing out that there are various alternatives to full subgraph isomorphism that have a lower complexity. For instance, as soon as an upper bound K is defined on the number of edges in a distinguishing graph, the problem loses its intractability (for relatively small K) and becomes solvable, in the worst case, in polynomial $\mathcal{O}(n^K)$ time, where n is number of edges in the graph G. Restricting the problem in such a way is rather harmless for our current purposes, as it prohibits the generation only of distinguishing descriptions with more than K properties, and for all practical purposes K can be small (referring expressions usually express a limited number of properties).

Defining an upper bound K, however, does have a disadvantage: We lose **completeness** (see van Deemter [2002]). In particular, the algorithm will fail for objects that can be uniquely described only with $K + 1$ (or more) edges. Of course, one could argue that in such cases objects should be distinguished using other means (e.g., by pointing). Nevertheless, it is worthwhile to look for classes of graphs for which the subgraph isomorphism problem can be solved more efficiently, without postulating upper bounds. For instance, if G and H are **planar** (simple) graphs the problem can be solved in time linear in the number of vertices of G (Eppstein 1999). Basically, a planar graph is one that can be drawn on a plane in such a way that there are no crossing edges (thus, for instance, the graph in Figure 3 is planar, as is any graph with only four vertices). In general, there is no a priori reason to assume that our scene representations will be planar. Yet every nonplanar graph can be modified into a closely related planar one. We briefly address planarization of scene graphs in the Appendix.

A final alternative is worth mentioning. The general approach to the problem of subgraph isomorphism detection assumes that both graphs are given on-line. For our current purposes, however, it may happen that the scene graph is fixed and known beforehand, and only the referring graph is unknown and given on-line. Messmer and Bunke (1995, 1998) describe a method that converts the known graph (or **model graph**, as they call it) into a decision tree. At run time, the input graph is classified by the decision tree, which detects subgraph isomorphisms. The disadvantage of this approach is that the decision tree may contain, in the worst case, an exponential number of nodes. But the main advantage is that the complexity of the new subgraph isomorphism algorithm is only quadratic in the number of vertices of the input referring graph. Note that with this approach we do not lose information from the scene graph, nor do we lose completeness.

In sum, the basic approach to subgraph isomorphisms is NP-complete, but there exist various reformulations of the problem that can be solved more efficiently. Deciding which (combination) of these is the most suitable in practice, however, is beyond the scope of this article. Finally, it is worth stressing that the NP-completeness is due to the presence of edges representing relations between different vertices. If we re-

strict the approach to properties (looping edges), testing for subgraph isomorphisms becomes trivial.

4. A Sketch of a Branch and Bound Generation Algorithm

In this section we give a high-level sketch of the graph-based generation algorithm. The algorithm (called **makeReferringExpression**) consists of two main components, a subgraph construction algorithm (called **findGraph**) and a subgraph isomorphism testing algorithm (called **matchGraphs**). For expository reasons we do not address optimization strategies (but see Section 6).

4.1 The Basic Idea

We assume that a scene graph $G = \langle V_G, E_G \rangle$ is given. The algorithm systematically tries all relevant subgraphs H of the scene graph G by starting with the subgraph containing only the vertex v (the target object) and expanding it recursively by trying to add edges from G that are adjacent to the subgraph H constructed up to that point. 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 (denoted as $G.neighbors(H)$). Formally:

$$G.neighbors(H) = \bigcup_{v \in V_H} \bigcup_{w \in V_G} E_G(v, w)$$

The algorithm returns the cheapest (least expensive) distinguishing subgraph H that refers to v, if such a distinguishing graph exists; otherwise it returns the undefined null graph $\perp$.

4.2 Cost Functions

We use cost functions to guide the search process and to give preference to some solutions over others. If $H = \langle V_H, E_H \rangle$ is a subgraph of G, then the costs of H, denoted as $cost(H)$, can be given by summing over the costs associated with the vertices and edges of H. Formally:

$$cost(H) = \sum_{v \in V_H} cost(v) + \sum_{e \in E_H} cost(e)$$

In fact, this is only one possible way to define a cost function. The only hard requirement cost functions have to fulfill is monotonicity. That is, adding an edge e to a graph G should never result in a graph cheaper than G. Formally:

$$\forall G' \subseteq G \; \forall e \in E_G \colon \; cost(G') \leq cost(G' + e)$$

The monotonicity assumption helps reduce the search space, since extensions of subgraphs with a cost greater than the best subgraph found up to that point can safely be ignored. Naturally, the costs of the undefined graph ($cost(\perp)$) are not defined. It is worth stressing that the cost function is global: It determines the costs of entire graphs. This implies that the cheapest distinguishing graph is not necessarily the smallest distinguishing graph; a graph consisting of two or more edges may be cheaper than a graph containing one expensive edge.

4.3 Worked Example

We now illustrate the algorithm with an example. Suppose the scene graph G is as given in Figure 3 and that we want to generate a referring expression for object d_1

```
makeReferringExpression(v) {
    bestGraph := ⊥;
    H := ⟨{v}, ∅⟩;
    return findGraph(v, bestGraph, H);
}

findGraph(v, bestGraph, H) {
    if [bestGraph ≠ ⊥ and cost(bestGraph) ≤ cost(H)]
    then return bestGraph
    fi;
    distractors := { n | n ∈ V_G ∧ matchGraphs(v, H, n, G) ∧ n ≠ v};
    if distractors = ∅ then return H fi;
    for each edge e ∈ G.neighbors(H) do
        I := findGraph(v, bestGraph, H + e);
        if [bestGraph = ⊥ or cost(I) ≤ cost(bestGraph)]
        then bestGraph := I
        fi;
    rof;
    return bestGraph;
}
```

Figure 6
Sketch of the main function (**makeReferringExpression**) and the subgraph construction function (**findGraph**).

in this graph. Let us assume for the sake of illustration that the cost function is defined in such a way that adding a vertex or an edge always costs one point. Thus, for each $v \in V_H$ and for each $e \in E_H$: $cost(v) = cost(e) = 1$. (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 6) with d_1 as parameter. In this function the variable *bestGraph* (for the best solution found up to that point) is initialized as the null graph (there is no best solution yet), and the variable H (for the distinguishing subgraph under construction) is initialized as the graph containing only vertex d_1 (i.e., (i) in Figure 7). Then the function **findGraph** (see also Figure 6) is called, with parameters d_1, *bestGraph*, and H.

To begin with, whether a first non-null *bestGraph* has been found is checked and, if one has, whether the costs of H (the graph under construction) are higher than the costs of the *bestGraph* found up to that point. If the costs of H are higher, it is not worth extending H since, due to the monotonicity constraint, it will never end up being cheaper than the current *bestGraph*. During the first iteration we have no non-null *bestGraph*, so we continue. Next the set of distractors is calculated. In terms of the graph perspective, this is the set of vertices in the scene graph G (other than the target vertex v) to which the graph H refers. It is easily seen that the initial value of H refers to every vertex in G. Hence, as one would expect, the initial set of distractors is $V_G \backslash \{d_1\} = \{d_2, d_3, d_4\}$. Then the current set of distractors is checked to determine whether it is empty. If it is, we have managed to find a distinguishing graph, which is subsequently stored in the variable *bestGraph*. In the first iteration, this is obviously not the case, and we continue, recursively trying to extend H by adding

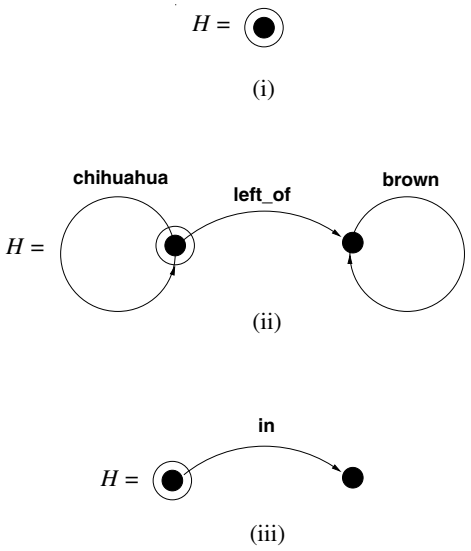

Figure 7
Three values for H in the generation process for d_1.

adjacent (neighboring) edges until either a distinguishing graph has been constructed (all distractors are ruled out) or the costs of H exceed the costs of the *bestGraph* found so far.

While *bestGraph* is still the null graph, the algorithm continues until H is a distinguishing graph. Which is the first distinguishing graph to be found (if more than one exists) depends on the order in which the adjacent edges are tried (see also Section 5.1). Suppose for the sake of argument that the first distinguishing graph to be found is (ii) in Figure 7. This graph is returned and stored in *bestGraph*. The costs associated with this graph are five points (two vertices and three edges). At this stage in the generation process only graphs with costs lower than five points are worth investigating. In fact, there are only a few distinguishing graphs that cost less than this. 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 7. That this distinguishing graph does not include type information does not necessarily mean that such information should not be realized. It means only that type information is, strictly speaking, not necessary to distinguish the intended referent from the distractors. We return to this issue in Section 5.2.

4.4 Subgraph Isomorphism Testing

Figure 8 sketches the part of the algorithm that 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 π such that $H \sqsubseteq_\pi G$ with $\pi.v = w$. The function **matchGraphs** first determines whether the looping edges starting from vertex v match those of w. If $E_H(v, v)$ is not a subset of $E_G(w, w)$ (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 try to expand it recursively. At each recursion step a fresh and as yet unmatched vertex y from V_H is selected that is adjacent to one of the vertices in the current domain of π (notated $dom(\pi)$). For each y we calculate the set Z of possible vertices in G to which y can be

```
matchGraphs(v, H, w, G) {
    if E_H(v,v) ⊈ E_G(w,w) then return false fi;
    π := {v ↦ w} ;
    Y := H.neighbors(v);
    return matchHelper(π, Y, G, H);
}

matchHelper(π, Y, G, H) {
    if | Dom (π)| = |H| then return true fi;
    if Y = ∅ then return false fi;
    choose a fresh, unmatched y from Y;
    Z := {z ∈ V_G | y might be matched to z };
    for each z ∈ Z do
       if z is a valid extension of the mapping
       then if matchHelper(π ∪ {y ↦ z}, Y, G, H)
            then return true
            fi;
       fi;
    rof;
    return false;
}
```

Figure 8
Sketch of the function testing for subgraph isomorphism (**matchGraphs**).

matched. This set consists of all the vertices in G that have the same looping edges as y and the same edges to and from other vertices in the domain of the current matching function π. Formally:

$$\begin{aligned} Z := \quad & \{z \mid z \in V_G \wedge E_H(y,y) \subseteq E_G(z,z) \wedge \\ & \quad \forall h \in H.neighbors(y) \cap dom(\pi) : \\ & \quad\quad [\ E_H(y,h) \subseteq E_G(z,\pi.h) \wedge E_H(h,y) \subseteq E_G(\pi.h,z)\ ] \\ & \} \end{aligned}$$

(The $H.neighbors(y)$ are the *vertices* in H that are adjacent to y, that is, those vertices that are connected to y by an edge.) The matching can now possibly be extended with $\pi.y = z$, for each $z \in Z$. The algorithm then branches over all these possibilities. Once a mapping π has been found that has exactly as many elements as H has vertices, we have found a subgraph isomorphism. If there are still unmatched vertices in H, or if all possible extensions with vertex y have been checked and no matching can be found, the test for subgraph isomorphism has failed.

4.5 A Note on the Implementation

The algorithm outlined in Figures 6 and 8 has been implemented in Java 2 (J2SE, version 1.4). The implemented version of the algorithm is actually more efficient than the sketch suggests, because various calculations need not be repeated in each iteration (the set of distractors and the set $G.neighbors(H)$, for example). In addition, the user has the possibility of specifying the cost function in whatever way he or she sees fit.

A full-fledged performance analysis of the current implementation is beyond the scope of this article. Such an analysis would be complicated by the fact that there

Table 1
Average times needed to find the cheapest distinguishing referring graphs for objects in seven test scene graphs of increasing complexity.

Test scene	Vertices	Edges	Average times
1	4	28	52 ms.
2	8	56	82 ms.
3	16	112	123 ms.
4	32	224	254 ms.
5	64	448	725 ms.
6	128	896	1,023 ms.
7	256	1,792	2,363 ms.

are many kinds of graphs (dense, sparse, (un)connected, etc.), and the performance results will vary with the properties of the scene graph. If the scene graph is fully connected, finding distinguishing graphs is much harder then when it is fully unconnected. Nevertheless, to give the reader some insight into the performance of the implementation, we applied it to seven test scene graphs of a specific form. The first is our running example: the graph in Figure 3. The other six are obtained by scaling up this graph and permutating (plus, if necessary, adding) properties to make sure that each object can be uniquely described. This implies that only the first test graph is fully connected. The other graphs consist of $\frac{n}{4}$ connected subgraphs, where n is the number of vertices in the graph. Thus, the bigger graphs are relatively less complex than the smaller ones. Each graph consists of $4n$ looping edges (representing properties) and $3n$ nonlooping edges (representing spatial relations), again with n the number of vertices.

The test was performed on a Windows 2000 PC with a 900 mHz AMD Athlon Processor and 128 Mb RAM. The results are given in Table 1. We measured the system time right before and right after the call of the main function **makeReferringExpression**. We computed the differences between these two times for a number of target objects from the scene graphs (4 and 8 objects for the first two graphs, 16 for the remaining ones). Note that this measurement does not include the time Java requires for initialization or background activities such as garbage collection. Table 1 shows that even for the larger graphs, the program is able to find minimal distinguishing graphs relatively quickly. The current implementation is a straightforward one (see also Section 6), so optimization, possibly in combination with heuristics, is likely to show further improvements in performance.

5. Cost Functions and Search Strategies

5.1 Full (Relational) Brevity Algorithm and Greedy Heuristics

The algorithm described in the previous section (with the uniform cost function assigning one point to each edge and vertex) 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. In this respect it is comparable to the Dale and Haddock (1991) algorithm, granted that here the problems with infinite recursions do not arise, since a particular edge is either present in a graph or not. Moreover, the approach is fully general and applies to n-ary relations (and relation/property combinations) as well.

It is worth noting that Dale's (1992) **greedy heuristic algorithm** (also discussed in Dale and Reiter [1995]) can be cast in the graph framework as well. In fact, this would give us a handle on the order in which different adjacent edges could be tried. The edges associated with the intended referent should be sorted on their descriptive power, which is inversely proportional to the number of occurrences of that particular edge in the scene graph. The algorithm then adds the most discriminating edge (i.e., the one removing most distractors) first. If there are various equally distinguishing edges, the cheapest one is added. This process is then repeated until a distinguishing graph is found. In fact, such a greedy strategy could be used to produce a first nonminimal distinguishing graph. Subsequently we could call **findGraph** with this graph as initial value of *bestGraph*, instead of the null graph $\perp$. In this way, we would be able to find minimal graphs more efficiently.

5.2 Incremental Algorithm

The characteristic properties of Dale and Reiter's (1995) incremental algorithm can be incorporated into the graph framework as follows. First, the list of preferred attributes can be modeled in terms of the cost function: All type edges should be cheaper than all other edges. In fact, (basic level, see below) type edges could be free. Moreover, the edges corresponding to absolute properties (color) should cost less than those corresponding to relative ones (size). This gives us exactly the effect of having a list of preferred attributes $\langle$ type, color, size $\rangle$. 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 included. The *incremental* nature of the incremental algorithm can be obtained by ordering edges with respect to their costs. Now the cheapest edges (i.e, those expressing type information) should be tried first, and more expensive edges should be tried later. In addition, the algorithm should terminate as soon as it has found a distinguishing graph. This *would* guarantee that bargain type loops are always included, and the algorithm would output (iii) from Figure 4 instead of (iii) from Figure 7 when applied to d_1 from Figure 3.

Another characteristic property of the original incremental algorithm (not discussed in section 2) is the use of a **subsumption hierarchy**. A chihuahua, for instance, can be referred to as either a chihuahua or a dog. The latter has a special status and is called the **basic level value** (see, e.g., Rosch [1978]). According to Dale and Reiter (1995) and Reiter (1990), human speakers have a general preference for basic level values and move to more specific (subsumed) values only if these are more informative. This notion of a subsumption hierarchy can be modeled using the cost function. For a given attribute, the basic level edges should be assigned the lowest costs, and those farthest away from the basic level edge should have the highest costs. This implies that adding an edge labeled dog is cheaper than adding an edge labeled chihuahua. Hence a chihuahua edge will be selected only when there are fewer (or less expensive) additional edges required to construct a distinguishing graph than would be the case for a graph including a dog edge. Note that (assuming that the scene representation is well defined) a distinguishing graph can never contain both a dog *and* a chihuahua edge, since there will always be a cheaper distinguishing graph omitting one of the two edges.

In sum, we can recast the incremental algorithm quite easily in terms of graphs. The original incremental algorithm operates only on properties (looped edges in graph terminology). Recall that when all edges in a scene graph are of the looping variety, testing for subgraph isomorphism becomes trivial. The graph-theoretical reformulation 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. Note that the use of a cost function to simulate subsumption hierarchies for properties carries over directly to relations; for instance, the costs of adding an 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 requires fewer (or less expensive) additional edges for the construction of a distinguishing graph. Another advantage of the way the graph-based algorithm models the list of preferred attributes is that finer-grained distinctions can be made than can with the incremental algorithm. In particular, we are not forced to say that values of the attribute *type* are *always* cheaper than values of the attribute *color*. Instead of assigning costs to attributes, we can assign costs to *values* of attributes. This gives us the freedom to assign edges labeled with a common color value (e.g., brown) a lower cost than edges labeled with obscure type values, such as Polish owczarek nizinny sheepdog. This implies that it will be cheaper to construct a distinguishing graph referring to an object using two cheap edges (*the brown dog*) than with one particularly expensive edge (*the Polish owczarek nizinny sheepdog*).

5.3 Aspects of Other Algorithms

Various aspects of other algorithms can be captured in the graph-based algorithm as well. To further illustrate the flexibility of the graph perspective, we briefly discuss two such aspects.

5.3.1 Plurals. Van Deemter's (2000) proposal to generate distinguishing plural descriptions (such as *the dogs*) can be modeled using graphs in the following way. Van Deemter's algorithm takes as input a set of target objects, which, in our case, translates into a set of vertices W from the scene graph ($W \subseteq V_G$). Now the algorithm tries to generate a vertex-graph pair (v, H) that uniquely refers to (W, G). The definition of "uniquely referring graphs" has to be generalized slightly to accommodate plurals. The constructed subgraph should refer to *each* of the vertices in the set W, but not to any of the vertices in the scene graph outside this set. Formally, (v, H) uniquely refers to (W, G) iff H is connected, and *for each* $w \in W$ there is a bijection π such that $H \sqsubseteq_\pi G$, with $\pi.v = w$ and there is no $w' \in G \backslash W$ such that (v, H) refers to (w', G). Observe that the singular case defined in Section 4 is obtained by restricting W to singleton sets. In this way, the basic algorithm can generate both singular and plural distinguishing descriptions.

5.3.2 Context and Salience. An object that has been mentioned in recent context is linguistically salient and hence can often be referred to using fewer properties; an animal that is first described as "the large black dog with the hanging ears" may subsequently be referred to using an anaphoric description such as "the dog." Krahmer and Theune (2002) model this phenomenon by assigning **salience weights** (sws) to objects. For this purpose they use a version of centering theory (Grosz, Joshi, and Weinstein 1995) augmented with a recency effect essentially due to Hajičová (1993). Krahmer and Theune then define the set of distractors as the set of objects with a salience weight higher than or equal to that of the target object. In terms of the graph-theoretical framework, this would go as follows. First, we assign salience weights to the vertices in the scene graph using the salience weights definition proposed by Krahmer and Theune. Subsequently, in the sketch of the basic algorithm (Figure 6), the set of distractors should be redefined as follows:

$$\{n \mid n \in V_G \wedge \textbf{matchGraphs}(v, H, n, G) \wedge n \neq v \wedge sw(n) \geq sw(v)\}$$

That is, the distractor set is restricted to those vertices n in the scene graph G that currently are at least as salient as the target object v. For target objects that are linguistically salient, this will typically lead to a reduction of the distractor set. Consequently, distinguishing graphs for these target objects will generally be smaller than those for nonsalient objects. Moreover, we will be able to find distinguishing graphs for a salient object v relatively fast, since we already have a distinguishing graph (constructed for the first definite reference to v) and we can use this graph as our initial value of *bestGraph*.

5.4 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 natural language processing (see, e.g., Langkilde and Knight [1998] and Malouf [2000] for partial answers). The approach proposed in this article makes it possible to combine graph reformulations of well-known rule-based generation algorithms with *stochastic* cost functions (the result resembles a Markov model). Such a cost function could be derived from a sufficiently large corpus. For instance, as a first approximation we could define the costs of adding an edge e in terms of the probability $P(e)$ that e occurs in a distinguishing description (estimated by counting occurrences):

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

Thus, properties that occur frequently are cheap; properties that are relatively rare are expensive. In this way, we would probably *derive* that polish owczarek nizinny sheepdog indeed costs more (and is thus less likely to be selected) than brown.

Even though this first approximation already has some interesting consequences, it is probably not enough to obtain a plausible and useful cost function. For instance, it is unlikely that the co-occurrence of edges is fully independent; a husky is likely to be white, and a chihuahua is not. Such dependencies are not modeled by the definition given above. In addition, properties referring to size such as small and large probably occur more often in a corpus than properties referring to colors such as brown or yellow, which at first sight appears to run counter to the earlier observation that speakers generally prefer absolute properties over relative ones. The reason for this, however, is probably that there are simply fewer ways to describe the size than there are to describe the color of objects. Searching for a more sophisticated method of defining stochastic cost functions is therefore an interesting line of future research.

6. Concluding Remarks and Future Research

In this article, we have presented a new approach to the content determination problem for referring expressions. We proposed to model scenes as labeled directed graphs, in which objects are represented as vertices and the properties and relations of these objects are represented as edges. The problem of finding a referring expression for an object is treated as finding a subgraph of the scene graph that is isomorphic to the intended referent but not to any other object. The theoretical complexity of this reformulation of the content determination problem is NP-complete, but there exist various restrictions (planar graphs, decision trees for fixed scene graphs, upper bound to the number of edges in a distinguishing graph) that have a polynomial complexity.

We have described a general and fully implemented algorithm, based on the subgraph isomorphism idea, consisting of two main functions: one that constructs referring graphs and one that tests for subgraph isomorphisms. Cost functions are used to

guide the search process and to give preference to some solutions over others. Optimization has not been the focus of this article, but we came across various heuristic strategies that would speed up the algorithm. For instance, we can try edges in the order determined by the cost function (from cheap to more expensive), and we can use a greedy algorithm to find a first distinguishing graph quickly. In general, one of the advantages of the graph perspective is that many efficient algorithms for dealing with graph structures are known. We can use those algorithms to formulate more efficient versions of the subgraph construction component (perhaps using the method of Tarjan [1972]; see also Sedgewick [1988]) and of the subgraph isomorphism testing component (e.g., using the aforementioned approach of Messmer and Bunke [1995, 1998]).

The graph perspective has a number of attractive properties. (1) By reformulating the content determination problem as a graph construction problem, we can directly apply the many techniques and algorithms for dealing with graph structures. (2) 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 mimic and extend various well-known algorithms from the literature (see also Krahmer, van Erk, and Verleg [2001]). (3) The generation of relational descriptions is straightforward; the problems that plague some other algorithms for the generation of relational descriptions do not arise. Moreover, the approach to relations proposed here is fully general: It applies to all *n*-ary relations, not just binary ones. (4) The use of cost functions paves the way for integrating statistical information directly into the generation process. In fact, performing experiments with various ways to estimate stochastic cost functions from corpora is one path for future research that we have identified.

Besides looking for graph-based optimizations and performing experiments with stochastic cost functions, there are three other lines for future research we would like to mention. The first concerns the construction of scene graphs. How should the decision be made as to which aspects of a scene to represent in the graph? Naturally, the algorithm can only refer to entities that are modeled in the scene graph, but representing every possible object in a single graph will lead to an explosion of edges and vertices. Perhaps some notion of **focus of attention** can be used to restrict the scene graph. It would also be interesting to look for automatic methods for the construction of scene graphs. We might use computer vision algorithms (see, e.g., Faugeras [1993]), which are often graph-based themselves, for this purpose. For example, Bauckhage et al. (1999) describe an assembly system in which computer vision is used to convert a workspace with various building blocks into a labeled directed scene graph. Note that this approach is also able to deal with dynamic scenes; it can track changes in the workspace (which is required for handling the assembly process).

Another issue that we have not discussed in much detail is linguistic realization. How should the information contained in a referring graph be expressed in natural language? So far, we have assumed that a distinguishing graph can simply be constructed first and subsequently fed into a realization engine. There may, however, be certain dependencies between content selection and realization (see, e.g., Horacek [1997] and Krahmer and Theune [2002]). One way to take these dependencies into account would be to reformulate the cost function in such a way that it promotes graphs that can easily be realized and punishes graphs that are more difficult to realize.

A final aspect of the graph model that deserves further investigation is based on the fact that we can look at a graph such as that in Figure 3 as a **Kripke model**. Kripke models are used in model-theoretic semantics for modal logics. The advantage of looking at graphs such as that in Figure 3 as Kripke models 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 vertex i, we are looking for that distinguishing formula φ that is true of ("at") i but not of any j different from i. One advantage of this logical perspective is that logical properties that are not covered by most generation algorithms (such as *not* having a certain property; see van Deemter [2002]) fit in very well with this perspective.

Appendix: Planarizing Scene Graphs

Planar graphs may be relevant for our current purposes, since subgraph isomorphism can be tested more efficiently on planar graphs than on arbitrary graphs. There are two ways in which a nonplanar graph G can be turned into a planar one G' (see Liebers [2001] for a recent overview of planarization algorithms): Either the graph G can be **pruned** (using vertex or edge deletion) or it can be **extended** (for instance, using vertex splitting or by inserting vertices at crossings). A disadvantage of the extension approach is that we lose the intuitive one-to-one correspondence between potential target objects and vertices in the scene graph, since the additional vertices only serve the purpose of planarizing the graph and do not represent objects in a scene. A disadvantage of the pruning approach is that we lose information. The presence of a cost function, however, is potentially very useful, since it allows us to avoid eliminating comparatively cheap (and thus more frequently selected) edges.

Here, for the sake of illustration, we briefly describe a **weighted greedy pruning algorithm** that turns an arbitrary scene graph $G = \langle V_G, E_G \rangle$ with n vertices and m edges into a planar graph $G' = \langle V_{G'}, E_{G'} \rangle$ with n vertices and at most m edges. We start from the graph $G' = \langle V_G, O_G \rangle$, where O_G is the set of looping edges from the scene graph, that is, $O_G = \bigcup_{v \in V_G} E_G(v, v)$. Next we order the remaining edges from the scene graph $R_G = E_G \backslash O_G$ with respect to their costs; the cheapest one comes first, the more expensive ones come later, in order of increasing expense. For each $e \in R_G$, we check whether $G' + e$ is planar (e.g., using the algorithm from Hopcroft and Tarjan [1974]). If it is, e is added to $E_{G'}$. The algorithm terminates when $R_G = \emptyset$. The result is a maximal planar subgraph G' of the scene graph G that differs from G only possibly in the deletion of certain relatively expensive nonlooping (relational) edges.

Acknowledgments

We would like to thank Dennis van Oort and Denis Gerritsen for their help in the implementation and Alexander Koller and Kees van Deemter for some very useful discussions. Thanks are also due to Paul Piwek, Mariët Theune, Ielka van der Sluis, and the anonymous reviewers for helpful comments on an earlier version of this article.

References

Appelt, Douglas E. 1985. Planning English referring expressions. *Artificial Intelligence*, 26:1–33.

Bateman, John. 1997. Enabling technology for multilingual natural language generation: The KPML development environment. *Natural Language Engineering*, 3:15–55.

Bateman, John. 1999. Using aggregation for selecting content when generating referring expressions. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL'99)*, University of Maryland.

Bauckhage, Christian, Jannik Fritsch, Franz Kummert, and Gerhard Sagerer. 1999. Towards a vision system for supervising assembly processes. In *Proceedings of the Symposium on*

Intelligent Robotic Systems (SIRS'99), pages 89–98.
Berge, Claude. 1985. *Graphs*. North-Holland, Amsterdam.
Blackburn, Patrick. 2000. Representation, reasoning, and relational structure: A hybrid logic manifesto. *Logic Journal of the IGPL*, 8(3):339–365.
Chartrand, Gary and Ortrud Oellermann. 1993. *Applied and Algorithmic Graph Theory*. McGraw-Hill, New York.
Cormen, Thomas H., Charles E. Leiserson, and Ronald L. Rivest. 1990. *Introduction to Algorithms*. MIT Press, Cambridge.
Dale, Robert. 1992. *Generating Referring Expressions: Constructing Descriptions in a Domain of Objects and Processes*. MIT Press, Cambridge.
Dale, Robert and Nicholas Haddock. 1991. Generating referring expressions involving relations. In *Proceedings of the Fifth Conference of the European Chapter of the Association for Computational Linguistics (EACL'91)*, pages 161–166, Berlin.
Dale, Robert and Ehud Reiter. 1995. Computational interpretations of the Gricean maxims in the generation of referring expressions. *Cognitive Science*, 18:233–263.
Elhaded, Michael and Jacques Robin. 1997. SURGE: A comprehensive plug-in syntactic realization component for text generation. Technical Report 96-03, Computer Science Department, Ben-Gurion University, Beer Sheva, Israel.
Eppstein, David. 1999. Subgraph isomorphism in planar graphs and related problems. *Journal of Graph Algorithms and Applications*, 3(3):1–27.
Faugeras, Olivier. 1993. *Three-Dimensional Computer Vision*. MIT Press, Cambridge.
Garey, Michael R. and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, New York.
Gibbons, Alan M. 1985. *Algorithmic Graph Theory*. Cambridge University Press, Cambridge.
Grosz, Barbara J., Aravind K. Joshi, and Scott Weinstein. 1995. Centering: A framework for modeling the local coherence of discourse. *Computational Linguistics*, 21(2):203–225.
Hajičová, Eva. 1993. *Issues of Sentence Structure and Discourse Patterns*. Charles University, Prague.
Hopcroft, John E. and Robert Tarjan. 1974. Efficient planarity testing. *Journal of the Association for Computing Machinery*, 21:549–568.
Horacek, Helmut. 1997. An algorithm for generating referential descriptions with flexible interfaces. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL'97)*, pages 206–213, Madrid.
Krahmer, Emiel and Mariët Theune. 1998. Context sensitive generation of descriptions. In *Proceedings of the Fifth International Conference on Spoken Language Processing (ICSLP'98)*, pages 1151–1154, Sydney.
Krahmer, Emiel and Mariët Theune. 2002. Efficient context-sensitive generation of descriptions. In Kees van Deemter and Rodger Kibble, editors, *Information Sharing: Givenness and Newness in Language Processing*. CSLI Publications, Stanford, California, pages 223–264.
Krahmer, Emiel, Sebastiaan van Erk, and André Verleg. 2001. A meta-algorithm for the generation of referring expressions. In *Proceedings of the Eighth European Workshop on Natural Language Generation (EWNLG'01)*, Toulouse.
Langkilde, Irene and Kevin Knight. 1998. The practical value of *n*-grams in generation. In *Proceedings of the Ninth International Workshop on Natural Language Generation (INLG'98)*, pages 248–255, Niagara-on-the-Lake, Ontario.
Liebers, Annegret. 2001. Planarizing graphs. *Journal of Graph Algorithms and Applications*, 5(1):1–74.
Malouf, Rob. 2000. The order of prenominal adjectives in natural language generation. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL'00)*, Hong Kong.
Messmer, Bruno T. and Horst Bunke. 1995. Subgraph isomorphism in polynomial time. Technical Report IAM 95-003, University of Bern, Institute of Computer Science and Applied Mathematics, Bern, Switzerland.
Messmer, Bruno T. and Horst Bunke. 1998. A new algorithm for error-tolerant subgraph isomorphism detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):493–504.
Pechmann, Thomas. 1989. Incremental speech production and referential overspecification. *Linguistics*, 27:98–110.
Read, Ronald C. and Derek G. Corneil. 1977. The graph isomorphism disease. *Journal of Graph Theory*, 1(1):339–363.
Reiter, Ehud. 1990. The computational complexity of avoiding conversational implicatures. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics (ACL'90)*,

pages 97–104.
Rosch, Eleanor. 1978. Principles of categorization. In E. Rosch and B. Lloyd, editors, *Cognition and Categorization*. Lawrence Erlbaum, Hillsdale, New Jersey, pages 27–48.
Sedgewick, Robert. 1988. *Algorithms*. Addison-Wesley, Reading, Massachusetts, second edition.
Stone, Matthew and Bonnie Webber. 1998. Textual economy through close coupling of syntax and semantics. In *Proceedings of the Ninth International Workshop on Natural Language Generation (INLG'98)*, pages 178–187, Niagara-on-the-Lake, Ontario.
Tarjan, Robert E. 1972. Depth-first search and linear time algorithms. *SIAM Journal on Computing*, 1(2):146–160.
Theune, Mariët. 2000. *From Data to Speech: Language Generation in Context*. Ph.D. thesis, Eindhoven University of Technology, Eindhoven, The Netherlands.
van Deemter, Kees. 2000. Generating vague descriptions. In *Proceedings of the First International Natural Language Generation Conference (INLG'00)*, Mitzpe Ramon.
van Deemter, Kees. 2002. Generating referring expressions: Boolean extensions of the incremental algorithm. *Computational Linguistics*, 28(1):37–52.
Wilson, Robin J. 1996. *Introduction to Graph Theory*. Longman, Harlow, England, fourth edition.