Using Set Constraints to generate Distinguishing Descriptions

Marilisa Amoia[1], Claire Gardent[2], and Stefan Thater[1]

[1] Computational linguistics
University of Saarbruecken
Germany
{amoia,stta}@coli.uni-sb.de
http://www.coli.uni-sb.de
[2] CNRS, Nancy
France
Claire.Gardent@loria.fr
http://www.loria.fr/ gardent

Abstract. Following on [Gar02] which shows that the descriptions output by van Deemter algorithm [vD] are not always linguistically appropriate, this paper shows how the task of generating a distinguishing description for a given target set given a certain context can be reformulated as a constraint satisfaction problem. It then investigates the practical performance of a constraint solver that was implemented in OZ for this CSP.

1 Introduction

Given some initial communicative goal (e.g., "describe event e_1"), the generation task consists in producing a natural language text which fulfills this goal (e.g., '"*The cat sleeps*").

The generation task is a complex choice problem where a good solution is a text which minimises the violations of the various interacting linguistic constraints. For instance, given some entity e_1 to be described, the choice of the words and of the linguistic form used to describe that entity will depend on the informational status of e_1 (e.g., whether it has been mentioned in the previous discourse and whether it is known to the hearer only or to both the speaker and the hearer), on the semantics of the other words used in the surrounding linguistic linguistic context and on world knowledge (information that follows from the context and/or from world knowledge can often be omitted). Success in fulfilling these constraints will affect the quality of the generated output that will, accordingly, be more or less effective in communicating the speaker's intention.

In this paper, we concentrate on a sub-task of the generation process namely, the task of generating distinguishing descriptions (DD). Given a set of entities (the universe) and a set of n-ary relations known to hold of these entities, this task consists in finding a set of relations which uniquely identifies a given subset (the target set) of the universe. So for instance, given a context with two white cats e_1 and e_2, one big (e_1) and the other small (e_2), a distinguishing description for e_1 will be { *white,cat,big* } which can be verbalised as e.g., *the big white cat.*

The problem of generating distinguishing descriptions is a well studied problem which has given rise to a number of algorithms ([DH91,DR95,Hor97,KvEV01]) among which the *incremental algorithm* described in [DR95] has become the standard reference. Originally conceived to describe single entities using exclusively conjunctions of positive relations, this algorithm was recently generalised by [vD] to describing sets of entities using conjunctions and disjunctions of negated and positive relations.

The purpose of this paper is to propose a constraint based alternative to the incremental algorithm proposed in [vD]. The motivation for this move is primarily linguistic as [Gar02] shows that van Deemter's algorithm generates linguistically inappropriate DD. The alternative proposal has a worse theoretical complexity however and is in fact, exponential in the number of relations holding of the entities to be described. This paper investigates the practical behaviour of our alternative proposal and compares it with that of the incremental algorithm.

2 The incremental approach

In essence, the algorithm proposed by Dale and Reiter [DR95] (henceforth D&R) incrementally builds a conjunction of properties by selecting properties that are true of the element to be described (henceforth, the *target element*) and by stopping either when this conjunction is true of only the target element (in case there exists a distinguishing description for the target element) or when there is no more property available (in case the target element has no distinguishing description).

$\mathcal{I}$: the universe;
P_e, the set of properties of e;
To generate the distinguishing description D_e, do:

- Initialise: $C := \mathcal{I}$, $D_e := \emptyset$.
- Check success:
  If $C = \{e\}$ **return** D_e
  elseif $P_e = \emptyset$ **then** fail
  else goto step 3.
- Choose property $p_i \in P_e$ which picks out the smallest set $C_i = C \cap \{x \mid p_i(x)\}$.
- Update: $D_e := D_e \cup \{p_i\}, C := C_i$, $P_e := P_e - \{p_i\}$. **goto** step 2.

Fig. 1. The D&R Algorithm.

Figure 1 gives a more detailed description of the basic algorithm: given a target entity e and the list P_e of its properties, the D&R algorithm starts with a distractor set C corresponding to the given universe and an empty description D_e. It then iterates through a loop where each step consists in selecting a property p_i that is true of e but not of every element in C (p_i is not true of all elements in the distractor sets and thereby can be used to reduce the size of the distractor set). The iteration stops either when the distractor set is the singleton set containing the target entity (the set of selected properties is then returned and the DD is the conjunction of these properties) or else when all properties of e have been used but the distractor set is not a singleton set (in this case there is no DD for e and the algorithm returns a failure).

A usual refinement of this basic algorithm concerns the order in which properties are selected. Roughly, properties are grouped into classes (e.g., type, shape, colour and size) and selected according to some predefined order. For instance, the algorithm might start first, by selecting a type (this ensures that a "nominal property" is included in the resulting DD i.e., a property that can be expressed by a noun), then a colour, then a shape etc.

$\mathcal{I}$: the domain;
$S \subseteq \mathcal{I}$, the set to be described;
P_S, the properties true of the set S ($\mathcal{P}_S^+ = \bigcap_{x \in S} \mathcal{P}_x^+$ with $\mathcal{P}_x^+$ the set of properties that are true of x);

To generate the distinguishing description D_S, do:

1. Initialise: $C := \mathcal{I}$, $D_S := \emptyset$.
2. Check success:
   If $C = S$ **return** D_S
   elseif $P_S = \emptyset$ **then** fail
   else goto step 3.
3. Choose property $p_i \in P_S$ s.t. $S \subseteq [[p_i]]$ and $C \not\subseteq [[p_i]]$
4. Update: $D_S := D_S \cup \{p_i\}, C := C \cap [[p_i]]$, $P_S := P_S - \{p_i\}$. **goto** step 2.

Fig. 2. Extending D&R Algorithm to sets of individuals.

In sum, the D&R algorithm builds distinguishing descriptions for *single* entities using *conjunctions* of properties. To extend its coverage and permit the description of *sets of entities* using also *disjunctive* and *negated* information, [vD] generalises the incremental algorithm as follows. First, the basic algorithm is generalised to sets of individuals in the obvious way (cf. Figure 2). That is, given a target set S, the algorithm starts as before with a distractor set C which initially is equal to the set of individuals present in the context. It then proceeds by incrementally selecting a property P that is true of the target set ($S \subseteq [[P]]$)

but not of all elements in the distractor set ($C \not\subseteq [[P]]$). Each selected property is thus used to simultaneously increment the description being built and to eliminate some distractors. Success occurs when the distractor set equals the target set. The result is a distinguishing description (DD, a description that is true only of the target set) which is the conjunction of properties selected to reach that state.

In order to generalise this algorithm to disjunctive and negative properties, van Deemter then adds one more level of incrementality, an incrementality over the length of the properties being used. First, literals are used i.e., atomic properties and their negation. If this fails, disjunctive properties of length two (i.e. with two literals) are used; then of length three etc.

Phase 1: Perform the extended D&R algorithm using all literals i.e., properties in $P_{+/-}$; if this is successful then stop, otherwise go to phase 2.

Phase 2: Perform the extended D&R algorithm using all properties of the form $P \vee P'$ with $P, P' \in P_{+/-}$; if this is successful then stop, otherwise go to phase 3.

etc.

Fig. 3. Extending D&R Algorithm to disjunctive properties

3 Problems with the incremental algorithm

As already noticed in [DR95], the incremental algorithm does not always generate a minimal description i.e., a description with fewest literals. For instance, suppose the ordering in which properties are selected is:

$$\textsc{Type} < \textsc{Material} < \textsc{Size} < \textsc{Colour}$$

Suppose further that the context is as follows and the entity to be described is $\{x_1\}$.

	Large	Small	Medium	Red	Green	Blue	Plastic	Paper	Cup
x_1	•			•			•		•
x_2		•		•			•		•
x_3		•		•				•	•
x_4			•	•				•	•
x_5	•				•			•	•
x_6	•					•		•	•
x_7	•					•	•		•

In that case, the DD produced by the D&R algorithm will be $\{plastic, large, red, cup\}$ whereas the shorter $\{large, red, cup\}$ suffices to uniquely identify the target.

More generally, whenever a subset P of the output description D is such that the extension of the conjunction formed out of the elements of P is a subset of the extension of some other property $p_i \in D \setminus P$, then p_i is redundant. So for instance, *plastic* is redundant in $\{plastic, large, red\}$ because the extension of *large*$\wedge$ *red* is included in the extension of *plastic*.

When generalising the D&R algorithm to take into account disjunctive properties, this problem becomes more acute. So for instance suppose the context is as illustrated in Figure 4 and the target set is $\{x_1, x_2\}$.

	pdt	secr	treasurer	board-member	member
x_1	•			•	•
x_2		•		•	•
x_3			•	•	•
x_4				•	•
x_5				•	•
x_6					•

Fig. 4. Epistemically redundant descriptions

To build a distinguishing description for the target set $\{x_1, x_2\}$, the incremental algorithm will first look for a property P in the set of literals such that (i) $\{x_1, x_2\}$ is in the extension of P and (ii) P is not true of all elements in the distractor set C (which at this stage is the whole universe i.e., $\{x_1, x_2, x_3, x_4, x_5, x_6\}$). Two literals satisfy these criteria: the property of being a board member and that of not being the treasurer. Suppose the incremental algorithm first selects the *board-member* property thereby reducing the distractor set to $\{x_1, x_2, x_3, x_4, x_5\}$. Then $\neg$*treasurer* is selected which restricts the distractor set to $\{x_1, x_2, x_4, x_5\}$. There is no other literal which could be used to further reduce the distractor set hence properties of the form $P \vee P'$ are used. At this stage, the algorithm might select the property *pdt*$\vee$*secr* whose intersection with the distractor set yields the target set $\{x_1, x_2\}$. Thus, the description produced is in this case: *board-member* $\wedge\neg$*treasurer* $\wedge$(*pdt* $\vee$ *secr*) which can be phrased as

(1) the president and the secretary who are board members and not treasurers

whereas the minimal distinguishing description (2) would be a much better output.

(2) the president and the secretary

One problem thus is that, although perfectly well formed minimal DDs might be available, the incremental algorithm may produce "epistemically redundant descriptions" i.e. descriptions which include information already entailed (through what we know) by some information present elsewhere in the description.

	White	Dog	Cow	Big	Small	Medium	Pitbut	Poodle	Holstein	Jersey
x_1	•									
x_2	•	•								
x_3	•		•							
x_4	•		•			•				
x_5	•		•			•				•
x_6	•		•		•				•	
x_7	•		•		•					
x_8	•	•		•						
x_9	•	•		•				•		
x_{10}	•	•		•			•			
x_{11}										

Fig. 5. Unnecessary long descriptions.

As shown in [Gar02], another aspect of the same problem is that the algorithm might also yield unnecessary long and ambiguous descriptions. So for instance, given the context represented in Figure 5 and the target set $\{x_5, x_6, x_9, x_{10}\}$, van Deemter's algorithm will yield the description

$$W \wedge (B \vee C) \wedge (H \vee \neg S) \wedge (J \vee \neg M)$$

which can be verbalised as e.g.,

(3) The white things that are big or a cow, a Holstein or not small, and a Jersey or not medium size

whereas the shortest solution is in this case a description involving a disjunction with four disjuncts namely

$$Pi \vee Po \vee \vee H \vee J$$

which can be verbalised as

(4) the Pitbul, the Pooddle, the Holstein and the Jersey

In short another problem with the extended incremental algorithm is that it might yield descriptions that are unnecessary ambiguous (because of the high number of logical connectives they contain) and in the extreme cases, incomprehensible.

4 An alternative based on set constraints

Because of the greedy strategy followed by the incremental algorithm, it is impossible to modify it so that it delivers *only* minimal solutions. However, it is probably possible to improve it so that it produces linguistically acceptable solutions *in most cases*. One possibility for instance, would be to structure the space formed by the boolean closure of the given literals through attributes and subsumption: given an attribute (e.g., TYPE), all properties formed by using disjunction, conjunction and negation on the set of literals given for that specific attribute would be computed and ordered in a subsumption lattice. One could then, as suggested in [DR95], go through the list of attributes in a fixed order and for each attribute pick the *most specific property* (atomic or complex) that is true of the target set and eliminates some distractors. Modified in that way, the incremental algorithm would for instance produce the desired result for example given in Figure 4.

Here we investigate another avenue of research which is suggested by the set theoretic analysis of van Deemter: we use set constraints as made available by constraint programming technology and redefine the problem of generating DD as a constraint satisfaction problem. The proposed approach only delivers *minimal descriptions* i.e., descriptions which use the smallest number of literals to uniquely identify the target set. The problems discussed above are thereby resolved since by definition, minimal solutions will never be redundant nor unnecessarily long and ambiguous.

As [DR95] shows, however, the problem of finding minimal distinguishing descriptions can be formulated as a set cover problem. Since set cover problems are known to be NP-hard [GJ79], an algorithm based on a naive *generate & test* strategy is likely to spawn an intractable search space. We therefore propose an alternative approach based on constraint-programming [MS98] which is typically applied to problems with high combinatorial complexity. The key idea underlying constraint-programming is to adopt a *propagate and distribute* strategy where distribution steps, which constitute search are put off as long as possible while propagation steps are performed first which draw inferences on the basis of efficient, deterministic inference rules and thereby prune the search space as much as possible.

We start in Section 4.1 by showing how the problem of finding conjunctive descriptions can be formulated declaratively as a constraint-satisfaction problem solvable by constraint programming. We then go on in Section 4.2 to present the constraint-system. In Section 4.3 we show how to extend the treatment of conjunctive descriptions to disjunctive descriptions. Finally, Section 4.4 shows how the treatment given so far extends to n-ary relations.

4.1 The Conjunctive Fragment

Given a target set S of individuals, a conjunctive distinguishing description is a formula of the form $\lambda x.\phi_1 \wedge \cdots \wedge \phi_k$ with ϕ_i being of the form $p(x)$ or $\neg p(x)$ for some predicate symbol p, which is satisfied by all and only the elements of S.

We represent such descriptions by pairs

$$D_S = \langle P_S^+, P_S^- \rangle$$

with P_S^+ being a set of positive predicate symbols i.e., symbols denoting properties that are true of all elements of the set S, and P_S^- being a set of negative predicate symbols i.e., symbols denoting properties that are false of all elements of S. The problem of finding a distinguishing description D_S for a given target set S thus consists in finding sets P_S^+ and P_S^- such that ($[\![p]\!]$ stands for the extension of p i.e., the set of individuals for which the property p holds):

1. for all $p \in P_S^+$ and for all $i \in S$, $i \in [\![p]\!]$,
2. for all $p \in P_S^-$ and for all $i \in S$, $i \notin [\![p]\!]$,
3. for all $i \notin S$ there is a $p \in P_S^+$ with $i \notin [\![p]\!]$ or there is a $p \in P_S^-$ with $i \in [\![p]\!]$.

The first two conditions ensure that the properties occurring in the resulting distinguishing description are either negative or positive properties of the target set (i.e., properties that holds either of all elements in the target set or of none). The third condition ensures that the distinguishing description holds only of the elements in the target set. It says that for any distractor d (i.e., all elements not in the target set), either there is a negative property of the target set which is true of d or there is a positive property of the target set which is not true of d.

The problem can be reformulated in terms of finite sets as follows: Let $\mathcal{I}$ be the set of individuals present in the context, let $\mathcal{P}$ be the set of properties and for every individual $i \in \mathcal{I}$, let $\mathcal{P}_i^+$ be the set of predicate symbols denoting properties that are true of i and let $\mathcal{P}_i^- = \mathcal{P} \setminus \mathcal{P}_i^+$ be the set of predicate symbols denoting properties that are false of i.

Further, we define

$$\mathcal{P}_S^+ = \cap\{\mathcal{P}_i^+ \mid i \in S\} \tag{1}$$

$$\mathcal{P}_S^- = \cap\{\mathcal{P}_i^- \mid i \in S\} \tag{2}$$

That is, $\mathcal{P}_S^+$ is the set of properties which are true of all elements in S while $\mathcal{P}_S^-$ is the set of properties which are false of all elements in S.

Now the three conditions that a distinguishing description must satisfy can be formulated as constraints on these sets as follows:

$$P_S^+ \subseteq \mathcal{P}_S^+ \tag{3}$$

$$P_S^- \subseteq \mathcal{P}_S^- \tag{4}$$

$$\forall i \notin S \ |(P_S^+ \setminus \mathcal{P}_i^+) \cup (P_S^- \cap \mathcal{P}_i^+)| > 0 \tag{5}$$

4.2 Solving the Constraints

The basic idea underlying our approach is to consider the equations (3)–(5) as a constraint satisfaction problem which can be solved directly by constraint-programming. A constraint satisfaction problem consists of a finite collection of variables taking values in finite domains together with a constraint ψ on these variables. A solution is an assignment that maps the variables to values such that the constraint ψ is satisfied.

In constraint-programming, solutions are computed incrementally as a sequence of improving approximations by alternating propagation and distribution steps. The basic idea is to distinguish *basic* and *non-basic* constraints. Basic constraints are "simple" constraints representing partial information that can be represented directly in the so-called constraint store. Non-basic constraints are implemented as concurrent agents (so-called propagators) that observe the constraint store and refine the partial informations by applying deterministic and efficient inference rules that derive new basic constraints.

Set Constraints. Basic constraints $\mathcal{B}$ represent partial informations of the value of a set variable S which are given by upper and lower bounds in conjunction with upper and lower bounds of the cardinality:

$$\mathcal{B} ::= D \subseteq S \subseteq D' \wedge n \leq |S| \leq n' \mid \mathcal{B}_1 \wedge \mathcal{B}_2 \mid \mathsf{false}$$

where D, D' are fixed finite sets and n, n' are fixed integers. These bounds can be refined by constraint propagation (see below).

Non-basic constraints include basic constraints and provide subset, union and intersection operations as constraints:

$$\mathcal{C} ::= \mathcal{B} \mid S_1 \cap S_2 = \emptyset \mid S_1 \subseteq S_2 \cup S_3 \mid \mathcal{C}_1 \wedge \mathcal{C}_2$$

All other desired set constraints can be expressed in terms of $\mathcal{C}$.

$$\begin{aligned} S_1 \subseteq S_2 &\equiv S_1 \subseteq S_2 \cup S_3 \wedge S_3 = \emptyset && \text{(inclusion)} \\ S_1 = S_2^C &\equiv S_1 \cap S_2 = \emptyset \wedge \Delta \subseteq S_1 \cup S_2 && \text{(complement)} \\ S_1 = S_2 \setminus S_3 &\equiv S_1 = S_2 \cap S_3^C && \text{(difference)} \end{aligned}$$

Propagation and Distribution. The operational semantics of non-basic constraints are given as inference rules (see e.g., [MÖ1] for a complete presentation). For instance, subset constraints $S_1 \subseteq S_2$ can infer that an integer n must be an element of S_2 as soon as the basic constraint $n \in S_1$ is derived. Similarly, $n \notin S_1$ is inferred as soon as $n \notin S_2$ is derived.

When no propagator can infer any new basic constraint but not all variables are determined, a non-deterministic search step (distribution) is applied. Depending on the chosen search strategy, a non-determined variable x is selected and the possible values are assigned to x. To ensure that solutions are searched for in increasing order of size, we distribute (i.e. make case distinctions) over the cardinality of the output description $|P_S^+ \cup P_S^-|$ starting with the lowest possible value. That is, first the algorithm will try to find a description $\langle P_S^+, P_S^- \rangle$ with cardinality one, then with cardinality two etc. The algorithm stops as soon as it finds a solution. In this way, the description output by the algorithm is guaranteed to always be the shortest possible description.

4.3 Extending the Algorithm to Disjunctive Properties

To extend our approach to disjunctive properties, the constraints used can be modified as follows. We represent a disjunctive description by a term

$$D_S = \langle D_{S_1}, \ldots, D_{S_{|S|}} \rangle$$

where D_{S_i} is a conjunctive distinguishing description for the set S_i with $1 \leq i \leq |S|$ and we require that

$$S = S_1 \cup \dots \cup S_{|S|} \tag{6}$$

That is, the algorithm looks for a tuple of sets such that their union $S_1 \cup \dots \cup S_{|S|}$ is the target set S and such that for each set S_i in that tuple there is a conjunctive distinguishing description D_{S_i}. The resulting term represents the disjunctive description $D_{S_1} \vee \ldots \vee D_{S_{|S|}}$ where each D_{S_i} is a conjunctive description i.e., a conjunction of literals.

However, the constraint system given in Section 4.2 needs to be extended because equations (1) and (2) cannot be expressed within this system. For conjunctive descriptions, this is not a problem, because the value of S in (1) and (2) is determined, hence the values of $\mathcal{P}_S^+$ and $\mathcal{P}_S^-$ can be calculated "offline" and need not be expressed as constraints. If we consider partitions, however, the set variables are possibly not determined and the equations must be expressed as constraints.

We therefore assume an additional "Select Intersection" constraint [Duc02] which allows (1) and (2) to be expressed directly as constraints.

As before solutions are searched for in increasing order of size (i.e., number of literals occurring in the description) by distributing over the cardinality of the resulting description.

4.4 Extending the Algorithm to n-ary relations

The set variables used in our constraints range over finite set of integers. This means that before constraints are applied, the algorithm encodes the objects being constrained—individuals and properties—into pairwise distinct integers. As a result, the proposed treatment straightforwardly generalises to n-ary relations. Just like the proposition $\mathit{red}(e_1)$ using the unary relation "*red*" can be encoded by an integer, so can the proposition $\mathit{on}(e_1, e_2)$ be encoded by several integers—one for *on* $e_1(e_2)$ and one for *on* $e_2(e_1)$.

5 Implementation and comparison with related algorithms

The constraint based approach presented in the preceding section has been implemented and its performance compared with two related algorithms: a *distribute-and-propagate* algorithm (which by comparison with the *propagate-and-distribute* sheds some light on the practical effect of propagation) and a basic version of van Deemter's algorithm, which permits comparison with the incremental approach. We now discuss both the implementation and the results of these comparisons.

5.1 Implementation

The constraint solver presented in section 4 was implemented using the concurrent constraint programming language Oz [Pro98] which supports set variables ranging over finite sets of integers and provides an efficient implementation of the associated constraint theory.

This constraint solver (which generates DDs i.e., sets of relations) has furthermore been integrated into the generation system InDiGen (`http://www.coli.uni-sb.de/cl/projects/indigen.html`) so that within this system, distinguishing descriptions are actually realised as definite descriptions i.e., nominal phrases with definite determiners (e.g., *the cat*). For more information on this part of the generation process, we refer the reader to [Gar02].

5.2 Propagate-and-distribute *v*. Distribute-and-Propagate

One motivation for adopting a constraint based approach when dealing with NP hard problems is that constraint programming is aimed at efficiently solving precisely this type of problems. The constraint programming language OZ for instance, deals with the combinatorial complexity inherent to these problems by following a *propagate-and-distribute* strategy which minimises the search by maximising propagation i.e., by maximising cheap deterministic inference. In effect, propagation eliminates choices (i.e., assignments of values to variables) which are incompatible with the problem constraints. Non deterministic choices (distribution) are made only when necessary i.e., when propagation is insufficient to resolve all ambiguities.

This first algorithmic comparison aims at demonstrating the practical effect of propagation both on the search space and on run times. This can be done very simply in Oz by inverting the processes and performing distribution before propagation. That is, instead of letting propagation prune the search space before making guesses about the solution, the program first uses distribution to guess a solution (i.e., it distributes on all the problem variables thereby determining a complete assignment) and then verifies that the resulting assignment satisfies the problem constraint. The only modification needed to obtain this effect involves the timing of distribution (i.e., when to make case distinctions). In all other respects, the two programs are identical.

By choosing the appropriate distribution strategy ("make a case distinction by picking the leftmost unspecified variable and assigning it the next higher cardinality") , the *Distribute-and-propagate* version of the algorithm is such that it follows a depth-first search and in effect, mimics the van Deemter algorithm sketched in section 2.

The result thus is a *generate-and-test* strategy which implements an extremely naive version of the incremental algorithm, one where all properties are tried out independent of whether or not they eliminate any distractors and are true of the target set. It is obviously not what van Deemter intends, but it is adequate to demonstrate the practical effects of propagation.

We compared the performance of the two algorithms using the following measures :

- The depth of the search tree
- The number of choice points

– The run time
– The first description produced

Let us start with the examples discussed in section 3. For the model given in Figure 4, we get the following results.

	Choice points	Tree Depth	Run Time	1st Solution
P&D	4	5	50 ms	The president and the secretary
D&P	106 438	23	22.95 s	The president and the secretary that are members and board members but not treasurers

The results clearly show the effect of propagation on the search space (tree depth and number of choice points) and improved overall performance (run times). For the model in Figure 5 on the other hand, we get:

	Choice points	Tree Depth	Run Time	1st Solution
P&D	3260	29	4.53 s	The Jersey, the Holstein, the Poodle and the Pitbul
D&P	-	-	-	Out of memory

This shows that propagation is insufficient when the number of disjuncts in the minimal solution grows. Here is a more detailed comparison of the two algorithms on that point.

As the table shows, increasing the number of disjuncts in the minimal solution quickly leads to a decrease in efficiency also for the *propagate-and-distribute* strategy. Again though performance is much better as with the incremental *generate-and-test* strategy.

Increasing number of properties. Comparing the two algorithms on models with an increasing number of properties, we get the following results.

<table>
<tr><th>Model</th><th></th><th>Propagate&Distribute</th><th>Distribute&Propagate</th></tr>
<tr><td rowspan="4"><table><tr><th></th><th>rabbit</th><th>man</th><th>hat</th><th>red</th></tr><tr><td>x_1</td><td>•</td><td></td><td></td><td>•</td></tr><tr><td>x_2</td><td>•</td><td></td><td></td><td>•</td></tr><tr><td>x_3</td><td>•</td><td></td><td></td><td></td></tr><tr><td>x_4</td><td></td><td></td><td>•</td><td></td></tr><tr><td>x_5</td><td></td><td>•</td><td></td><td></td></tr></table></td><td>CP</td><td>0</td><td>221</td></tr>
<tr><td>D</td><td>1</td><td>9</td></tr>
<tr><td>RT</td><td>0 ms</td><td>40 ms</td></tr>
<tr><td>$D_{[x1x2]}$</td><td>the rabbit that is not red</td><td>the rabbit that is not red</td></tr>
<tr><td rowspan="4"><table><tr><th></th><th>rabbit</th><th>man</th><th>hat</th><th>red</th><th>big</th></tr><tr><td>x_1</td><td>•</td><td></td><td></td><td>•</td><td>•</td></tr><tr><td>x_2</td><td>•</td><td></td><td></td><td>•</td><td>•</td></tr><tr><td>x_3</td><td>•</td><td></td><td></td><td></td><td></td></tr><tr><td>x_4</td><td></td><td></td><td>•</td><td></td><td></td></tr><tr><td>x_5</td><td></td><td>•</td><td></td><td></td><td></td></tr></table></td><td>CP</td><td>3</td><td>950</td></tr>
<tr><td>D</td><td>4</td><td>11</td></tr>
<tr><td>RT</td><td>10 ms</td><td>190 ms</td></tr>
<tr><td>$D_{[x3]}$</td><td>the rabbit that is not big</td><td>the rabbit that is not red and not big</td></tr>
<tr><td rowspan="4"><table><tr><th></th><th>rabbit</th><th>man</th><th>hat</th><th>red</th><th>big</th><th>large</th></tr><tr><td>x_1</td><td>•</td><td></td><td></td><td>•</td><td>•</td><td>•</td></tr><tr><td>x_2</td><td>•</td><td></td><td></td><td>•</td><td>•</td><td>•</td></tr><tr><td>x_3</td><td>•</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>x_4</td><td></td><td></td><td>•</td><td></td><td></td><td></td></tr><tr><td>x_5</td><td></td><td>•</td><td></td><td></td><td></td><td></td></tr></table></td><td>CP</td><td>3</td><td>3943</td></tr>
<tr><td>D</td><td>4</td><td>13</td></tr>
<tr><td>RT</td><td>10 ms</td><td>760 ms</td></tr>
<tr><td>$D_{[x3]}$</td><td>the rabbit that is not large</td><td>the rabbit that is not red,
not big and not large</td></tr>
</table>

<table>
<tr><th>Model</th><th></th><th>Propagate&Distribute</th><th>Distribute& Propagate</th></tr>
<tr><td rowspan="4"><table><tr><td></td><td>rabbit</td><td>cat</td></tr><tr><td>x_1</td><td>•</td><td></td></tr><tr><td>x_2</td><td></td><td>•</td></tr></table></td><td>CP</td><td>2</td><td>130</td></tr>
<tr><td>D</td><td>3</td><td>11</td></tr>
<tr><td>RT</td><td>10 ms</td><td>130 ms</td></tr>
<tr><td>$D_{[x1x2]}$</td><td>the rabbit and the cat</td><td>the rabbit and the cat</td></tr>
<tr><td rowspan="4"><table><tr><td></td><td>rabbit</td><td>red</td><td>cat</td></tr><tr><td>x_1</td><td>•</td><td>•</td><td></td></tr><tr><td>x_2</td><td></td><td></td><td>•</td></tr></table></td><td>CP</td><td>4</td><td>1024</td></tr>
<tr><td>D</td><td>5</td><td>15</td></tr>
<tr><td>RT</td><td>10 ms</td><td>320 ms</td></tr>
<tr><td>$D_{[x1x2]}$</td><td>the rabbit and the cat</td><td>the red rabbit and the cat</td></tr>
<tr><td rowspan="4"><table><tr><td></td><td>rabbit</td><td>cat</td></tr><tr><td>x_1</td><td>•</td><td></td></tr><tr><td>x_2</td><td></td><td>•</td></tr><tr><td>x_3</td><td>•</td><td></td></tr></table></td><td>CP</td><td>6</td><td>692</td></tr>
<tr><td>D</td><td>6</td><td>14</td></tr>
<tr><td>RT</td><td>10 ms</td><td>200 ms</td></tr>
<tr><td>$D_{[x1x2x3]}$</td><td>the rabbits and the cat</td><td>the rabbits and the cat</td></tr>
<tr><td rowspan="4"><table><tr><td></td><td>rabbit</td><td>cat</td><td>dog</td></tr><tr><td>x_1</td><td>•</td><td></td><td></td></tr><tr><td>x_2</td><td></td><td>•</td><td></td></tr><tr><td>x_3</td><td></td><td></td><td>•</td></tr></table></td><td>CP</td><td>8</td><td>138 308</td></tr>
<tr><td>D</td><td>6</td><td>23</td></tr>
<tr><td>RT</td><td>20 ms</td><td>33.73s</td></tr>
<tr><td>$D_{[x1x2x3]}$</td><td>the rabbit, the cat and the dog</td><td>the rabbit, the cat and the dog</td></tr>
<tr><td rowspan="4"><table><tr><td></td><td>rabbit</td><td>cat</td><td>dog</td><td>cow</td></tr><tr><td>x_1</td><td>•</td><td></td><td></td><td></td></tr><tr><td>x_2</td><td></td><td>•</td><td></td><td></td></tr><tr><td>x_3</td><td></td><td></td><td>•</td><td></td></tr><tr><td>x_4</td><td></td><td></td><td></td><td>•</td></tr></table></td><td>CP</td><td>32</td><td>Out of memory</td></tr>
<tr><td>D</td><td>11</td><td></td></tr>
<tr><td>RT</td><td>40 ms</td><td></td></tr>
<tr><td>$D_{[x1x2x3x4]}$</td><td>the rabbit, the cat, the dog and the cow</td><td></td></tr>
</table>

Again the results show that in practice, the *propagate-and-distribute* strategy deals more efficiently with the combinatorics.

5.3 Propagate-and-distribute *v.* Incremental

The *Distribute-and-propagate* algorithm discussed above is a very inefficient implementation of the incremental strategy for two reasons. First, as already mentioned, it tries out every property independent of whether or not this property is true of the target set. Second, the constraints used to check success are much more complex than needed: they check subset relations, intersection etc. when all that is needed (in the incremental strategy) is to check whether the current contrast set is a singleton set. We therefore implemented a third algorithm which strictly embodies the incremental strategy proposed by Van Deemter and compared it with our *propagate-and-distribute* proposal. The results are unsurprising. They show that the greedy strategy performs efficiently in all of the cases discussed in this paper (run times between 0 and 5 ms) but that the quality of the output is less than optimal with many output descriptions being overly redundant.

6 Conclusion

Following on [Gar02] which shows that the descriptions output by van Deemter algorithm are not always linguistically appropriate, this paper shows how the task of generating a distinguishing description can be reformulated as a constraint satisfaction problem.

The essential differences between the two proposals are both linguistic and computational. Whereas the incremental algorithm uses a greedy polynomial strategy but does not always deliver the shortest distinguishing description, the constraint based approach is potentially exponential but always delivers the shortest distinguishing description.

This paper investigates the practical performance of a corresponding constraint solver implemented in OZ and identifies some of its practical limitations. It shows in particular that efficiency rapidly decreases as the number of disjuncts in the minimal solution grows; and that the proof-of-concept prototype we developed cannot deal with large databases.

Accordingly, future work will concentrate on defining additional constraints and heuristics and on integrating them in the CSP formulated here. Performance is not the only decisive issue here and linguistic considerations are also at play. Indeed although we have (somewhat simplistically) assumed that an optimal solution is a minimal description (i.e., a description containing the smallest number of literals) in reality, other factors affect the adequacy of a description such as the use of appropriate categorisations, salient properties and simple structures. A more realistic approach to the problem then would be one that would identify and integrate these factors in the proposed model.

Acknowledgements

We thank Denys Duchier for implementing the basic constraint solver on which this paper is based. We also gratefully acknowledge the financial support of the Conseil Régional de Lorraine and of the Deutsche Forschungsgemeinschaft, Projet InDiGen in the Schwerpunktsprogramm “Sprachproduktion”.

References

[DH91] R. Dale and N. Haddock. Content determination in the generation of referring expressions. *Computational Intelligence*, 7(4):252–265, 1991.

[DR95] R. Dale and E. Reiter. Computational interpretations of the gricean maxims in the generation of referring expressions. *Cognitive Science*, 18:233–263, 1995.

[Duc02] D. Duchier. Configuration of labeled trees under lexicalized constraints and principles. To appear in the Journal of Language and Computation, 2002.

[Gar02] C. Gardent. Generating minimal definite descriptions. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, 2002.

[GJ79] W. Garey and D. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness*. W.H.Freeman, San Francisco, 1979.

[Hor97] H. Horacek. An algorithm for generating referential descriptions with flexible interfaces. In *Proceedings of the 35^{th} Annual Meeting of the Association for Computational Linguistics)*, pages 206–213, Madrid, 1997.

[KvEV01] E. Krahmer, S. van Eerk, and André Verleg. A meta-algorithm for the generation of referring expressions. In *Proceedings of the 8th European Workshop on Natural Language Generation*, Toulouse, 2001.

[M0̈1] T. Müller. *Constraint Propagation in Mozart.* PhD thesis, Naturwissenschaftlich-Technische Fakultät I, Fachrichtung Informatik, Saarbrücken, Germany, 2001.

[MS98] M. Mariott and P. J. Stuckey. *Programming with Constraints.* Kluwer Academic Publishers, 1998.

[Pro98] Programming Systems Lab Saarbrücken, 1998. Oz Webpage: `http://www.ps.uni-sb.de/oz/`.

[vD] K. van Deemter. Generating Referring Expressions: Boolean Extensions of the Incremental Algorithm. To appear in Computational Linguistics.