Discourse Planning as an Optimization Process

Ingrid Zukerman and Richard McConachy
Department of Computer Science, Monash University
Clayton, VICTORIA 3168, AUSTRALIA
email: {ingrid,ricky}@bruce.cs.monash.edu.au
phone: +61 3 905-5202 fax: +61 3 905-5146

Abstract. Discourse planning systems developed to date apply local considerations in order to generate an initial presentation that achieves a given communicative goal. However, they lack a global criterion for selecting among alternative presentations. In this paper, we cast the problem of planning discourse as an optimization problem, which allows the definition of a global optimization criterion. In particular, we consider two such criteria: (1) generating the most concise discourse, and (2) generating the 'shallowest' discourse, i.e., discourse that requires the least prerequisite information. These criteria are embodied in a discourse planning mechanism which considers the following factors: (1) the effect of a user's inferences from planned utterances on his/her beliefs, (2) the amount of prerequisite information a user requires to understand an utterance, and (3) the amount of information that must be included in referring expressions which identify the concepts mentioned in an utterance. This mechanism is part of a discourse planning system called WISHFUL-II which generates explanations about concepts in technical domains.

1 Introduction

Schema-based Natural Language Generation (NLG) systems, e.g., [Weiner, 1980; McKeown, 1985; Paris, 1988], determine the information to be presented based on common patterns of discourse. Goal-based planners, e.g., [Moore and Swartout, 1989; Cawsey, 1990], select a discourse operator if its prescribed effect matches a given communicative goal. If there is more than one such operator, the operator whose prerequisite information is believed by the user is preferred. However, if all the candidate operators require the generation of discourse that conveys some prerequisite information, the selection process is either random or the system designer determines in advance which operators should be preferred.

In this paper, we cast the problem of planning discourse that achieves a given communicative goal as an application of an optimization algorithm. This approach supports the definition of different optimization objectives, such as generating (1) the most concise discourse; (2) the 'shallowest' discourse, i.e., discourse that requires the least amount of prerequisite information; or (3) the most concrete discourse, i.e., discourse with the most examples. The resulting mechanism is part of a discourse planning system called WISHFUL-II, which is a descendant of the WISHFUL system described in [Zukerman and McConachy, 1993a].

Table 1 illustrates the discourse generated by our system using the concise and the shallow optimization objectives.

Table 1. Sample Concise and Shallow 'Wallaby' Discourse

Concise Discourse	Shallow Discourse
Wallabies have a pouch, which is like a pocket. They are like kangaroos, but they are 3 ft. tall.	Wallabies are Marsupials and they come from Australia. They hop and they are 3 ft. tall. Wally is a wallaby.

These texts were generated in order to convey the attributes *type, habitat, body parts, height* and *transportation mode* of the concept Wallaby to a user who owns a toy wallaby called Wally, and knows something about kangaroos, but is not familiar with the term pouch.

The concise discourse conveys most of the intended information by means of a Simile between wallabies and kangaroos. The Simile also yields the erroneous inference that wallabies are the same height as kangaroos. To contradict this inference, the system asserts that wallabies are 3 ft. tall. Since the user does not know that kangaroos have a pouch, this is asserted, and since the user does not know what a pouch is, information which evokes this concept is presented. The shallow discourse, on the other hand, uses Wally (the toy wallaby) to convey the body parts of a wallaby without naming them explicitly. This information is complemented by Assertions about a wallaby's type, habitat, height and transportation mode.

In the next section, we present an overview of WISHFUL-II. In Section 3, we describe the discourse planning mechanism. We then discuss the results we have obtained, and present concluding remarks.

2 Overview of the System

WISHFUL-II receives as input a *concept* to be conveyed, e.g., Wallaby, a list of *aspects* that must be conveyed about this concept, e.g., *habitat* and *body parts*, and a desired level of expertise the user should attain as a result of the presentation.

WISHFUL-II was used to generate descriptive discourse in various technical domains, such as chemistry, high-school algebra, animal taxonomy and Lisp. It produces multi-sentence paragraphs of connected English text. The discourse planning mechanism, which is the focus of this paper, generates a set of *Rhetorical Devices (RDs)*, where each RD is a rhetorical action, such as Assert, Negate or Instantiate, applied to a proposition. This set of RDs is optimal with respect to a given optimization criterion, e.g., conciseness or depth.

The following steps are performed by WISHFUL-II.

1. **Content Selection** – WISHFUL-II consults a model of the user's beliefs in order to determine which propositions

must be presented to convey the given aspects of a concept. This step selects propositions about which the user has misconceptions, propositions that are unknown to the user, and propositions that are not believed by the user to the extent demanded by the desired level of expertise.

2. **Generation of the Optimal Set of RDs** – WISHFUL-II searches for a set of RDs that conveys the propositions generated in the previous step while satisfying a given optimization objective. The process of generating candidate sets of RDs considers the following factors: (1) the effect of inferences from an RD on a user's beliefs; (2) the amount of prerequisite information required by the user to understand the concepts in an RD; and (3) the amount of information to be included in referring expressions which identify the concepts in an RD.
3. **Discourse Structuring** – A discourse structuring mechanism extracts discourse relations and constraints from the set of RDs generated in Step 2. It then generates an ordered sequence of the RDs in this set, where the strongest relations between the RDs are represented and no constraints are violated. Where necessary, the RDs in this sequence are interleaved with conjunctive expressions that signal the relationship between them [Zukerman and McConachy, 1993b].
4. **Generation of Anaphoric Referring Expressions** – Anaphoric referring expressions are generated for RDs that refer to a concept in focus. This process follows the organization of the discourse, since the appropriate use of anaphora depends on the structure of the discourse.
5. **Discourse Realization** – The resulting sequence of RDs is realized in English by means of the Functional Unification Grammar described in [Elhadad, 1992].

3 Generating the Optimal Set of RDs

The main stage of the optimization procedure consists of generating alternative sets of RDs that can convey a set of propositions. The first step in this stage consists of generating candidate RDs that can convey each proposition separately. To this effect, WISHFUL-II reasons from the propositions determined in the content selection step to the RDs that may be used to convey these propositions. This reasoning mechanism has been widely used in NLG systems, e.g., [Moore and Swartout, 1989; Cawsey, 1990].

The process of generating a set of RDs that can convey a set of propositions is not a straightforward extension of the process of generating candidate RDs that can convey each proposition separately. This is due to the following reasons: (1) an inference from an RD generated to convey a proposition p_i may undermine the effect of an RD generated to convey a proposition p_j; and (2) an RD generated to convey a proposition p_i may be made obsolete by an RD which was generated to convey another proposition, but from which the user can infer p_i. Further, it is not sufficient to propose a single set of RDs that can convey a set of propositions, because a set of RDs that initially appears to be promising may require a large number of RDs to convey its prerequisite information or to identify the concepts mentioned in it. Thus, after generating the RDs that can convey each of the intended propositions separately, the optimization procedure must consider concurrently the following inter-related factors in order to generate candidate sets of RDs that can convey all the intended propositions: (1) the effect of the RDs in a set on a user's beliefs, (2) the prerequisite information that the user must know in order to understand these RDs, and (3) the referring expressions required to identify the concepts mentioned in these RDs.

Owing to the interactions between the RDs in a set, the problems of generating the most concise set of RDs and generating the shallowest set of RDs are NP-hard[1]. Since this level of complexity is likely to be maintained for other optimization objectives, we have chosen a weak search procedure for the implementation of the optimization process.

In the following sections, we describe the optimization process as an application of the Graphsearch algorithm [Nilsson, 1980], and discuss the implementation of the main steps of this algorithm.

3.1 The Basic Optimization Procedure

Our optimization procedure, *Optimize-RDs*, receives as input the set of propositions generated in the content selection step of WISHFUL-II. It implements a simplified version of the Graphsearch algorithm [Nilsson, 1980] to generate a set of RDs that conveys these propositions and satisfies a given optimization criterion. The discourse planning considerations are incorporated during the expansion stage and the evaluation stage of Graphsearch.

The expansion stage of our procedure activates algorithm *Expand-sets-of-RDs*, which generates alternative *minimally sufficient* sets of RDs that can convey a set of intended propositions (Step 5 in procedure *Optimize-RDs*). A set of RDs is minimally sufficient if the removal of any RD causes the set to stop conveying the intended information. Note that a minimally sufficient set of RDs is not necessarily a minimal set of RDs. For example, both of the alternatives in Table 1 are composed of minimally sufficient sets of RDs. In this stage, the procedure also determines which prerequisite propositions must be known to the user to enable a set of RDs to convey the intended propositions, and which referring expressions are required to identify the concepts in a set of RDs. During the evaluation stage, the procedure ranks each set of RDs in relation to the other candidates, and prunes redundant RDs (Step 7). Both the ranking process and the pruning process consider the extent to which a set of RDs is likely to satisfy a given optimization criterion.

Algorithm **Optimize-RDs**({*propositions*})

1. Create a search graph G consisting solely of a start node s which contains {*propositions*}. Put s in a list called *OPEN*.
2. LOOP: If *OPEN* is empty, exit with failure.
3. Select the first node in *OPEN*, remove it from *OPEN*. Call this node n.
4. If n does not require prerequisite information or referring expressions, then exit successfully (n is a goal node).
5. **Expansion:** $M \leftarrow$ *Expand-sets-of-RDs*(n).
   Install the sets of RDs in M as successors of n in G.

[1] Finding a concise set of RDs that conveys a set of propositions reduces to the Minimum Covering problem, and finding a shallow set of RDs that conveys a set of propositions reduces to Satisfiability [Garey and Johnson, 1979].

6. Add the successors of node n to *OPEN*.
7. Evaluation: Reorder the nodes in *OPEN* and prune redundant nodes according to the given optimization criterion.
8. Go LOOP.

3.2 Expanding Sets of RDs

Procedure *Expand-sets-of-RDs* receives as input a node to be expanded, and returns all the minimally sufficient sets of RDs that convey the set of propositions in this node, accompanied by their respective prerequisite propositions and referring expressions. We compute *all* the minimally sufficient sets of RDs, rather than just the minimal set of RDs, because a set of RDs that initially appears to be promising may require a large number of RDs in order to convey its prerequisite information or to identify the concepts in it.

Algorithm **Expand-sets-of-RDs**(n)

1. Determine RDs that increase a user's belief in each proposition in node n. (Not all the RDs generated in this step are capable of conveying an intended proposition by themselves, but they may be able to do so in combination with other RDs.) **(Section 3.2.1)**
2. Use these RDs to construct minimally sufficient sets of RDs that convey all the propositions in n jointly. Put these sets of RDs in $\{\mathcal{MRD}\}$. **(Section 3.2.2)**
3. Determine the prerequisite propositions required by each set of RDs in $\{\mathcal{MRD}\}$ so that the user can understand it. **(Section 3.2.3)**
4. Determine referring expressions which evoke the concepts in each set of RDs in $\{\mathcal{MRD}\}$. **(Section 3.2.4)**

The output of *Expand-sets-of-RDs* takes the form of a set of *RD-Graphs*. An RD-Graph is a directed graph that contains the following components: (1) the set of propositions to be conveyed ($p_1, \ldots, p_n$ in Figure 1); (2) a minimally sufficient set of RDs ($RD_1, \ldots, RD_m$); (3) the effect of the inferences from the RDs in this set on the intended propositions, and possibly on other (unintended) propositions (labelled $w_{i,j}$); (4) the prerequisite propositions that enable these RDs to succeed ($p'_1, \ldots, p'_k$); (5) the relations between the prerequisite propositions and the main RDs (in thick lines); (6) the sets of RDs that evoke concepts in the main RDs ($\{RD^{m+1}\}, \ldots, \{RD^{m+l}\}$); and (7) the relations between the evocative sets of RDs and the main RDs (labelled $v_{m+i,j}$). The main set of RDs and the relations between the RDs in this set and the propositions to be conveyed are generated in Step 2 above. The weight $w_{i,j}$ contains information about the effect of RD_i on the user's belief in proposition p_j. The prerequisite propositions are generated in Step 3, and the evocative sets of RDs and their corresponding links are produced in Step 4.

3.2.1 Determining RDs

Given a list of propositions to be conveyed, $\{p\}$, in this step we propose RDs that can increase a user's belief in each of these propositions. To this effect, for each proposition $p_i \in \{p\}$ we first consider the following RDs: Assertion (A), Negation (N), Instantiation (I) and Simile (S),

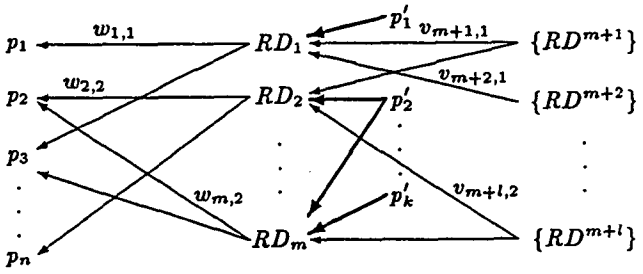

Figure 1. An RD-Graph

where different Instantiations and Similes may be generated for each proposition. For example, the proposition [Bracket-Simplification step-1 +/−] may be instantiated with respect to Numbers, e.g., $2(3+5) = 2 \times 8$, and to Like Terms, e.g., $2(3x+5x) = 2 \times 8x$. Those RDs that increase a user's belief in p_i are then put in a list called *RD-list*(p_i). Next, for each proposition p_i, we consider the RDs in *RD-list*(p_i). If an inference from any of these RDs increases a user's belief in a proposition $p_j \neq p_i$, this RD is added to *RD-list*(p_j). In addition, if any of the generated RDs yields an incorrect belief with respect to a proposition that is not in $\{p\}$, this proposition is added to $\{p\}$, and the process of determining RDs is repeated for this proposition in conjunction with the other propositions in $\{p\}$. This is necessary because RDs that are used to convey this new proposition could affect other propositions previously in $\{p\}$ and vice versa. This process stops when no incorrect beliefs are inferred.

This process is implemented in algorithm *Determine-RDs*, which receives three input parameters: (1) the propositions for which RDs were generated in the previous recursive call to *Determine-RDs*, (2) the propositions to be considered in the current activation of *Determine-RDs*, and (3) the *RD-list* generated in the previous recursive calls. Its initial activation is *Determine-RDs*(*nil*,{p}, *nil*), and its output is *RD-list*.

Algorithm *Determine-RDs*({*oldp*},{*newp*},*RD-list*)

1. **Backward reasoning:**
   For each proposition $p_i \in \{newp\}$ do:
   (a) Consider the following RDs: Assert(p_i), Negate($\neg p_i$), Instantiate(p_i,I) and Say-Simile(O_i,O), where the Instantiation is performed with respect to an instance I, and the Simile is performed between an object O_i, which is the subject of proposition p_i, and another object O[2]. (Note that several instances I and objects O may be used to generate different Instantiations and Similes, respectively.)
   (b) Assign to *RD-list*(p_i) the RDs that increase the user's belief in p_i.
2. **Forward reasoning:**
   (a) For each proposition $p_i \in \{newp\}$ determine whether any RD in *RD-list*(p_i) supports other propositions in $\{oldp\} \bigcup \{newp\}$. If so, add this RD to the *RD-lists* of these propositions.

[2] Other RDs that may be generated involve subclass or superclass concepts of the target concept in an intended proposition. However, the generation of these RDs has not been incorporated into WISHFUL-II yet.

(b) For each proposition $p_i \in \{oldp\}$ determine whether any RD in *RD-list*(p_i) supports propositions in $\{newp\}$. If so, add this RD to the *RD-lists* of these propositions.
(c) Append the propositions in $\{newp\}$ to $\{oldp\}$.
(d) If any RD used to convey a proposition $p_i \in \{newp\}$ yields incorrect beliefs, then
  i. Assign to $\{newp\}$ the propositions which contradict these beliefs.
  ii. Assign to *RD-list* the result returned by Determine-RDs($\{oldp\}$,$\{newp\}$,*RD-list*)
3. Return(*RD-list*)

To illustrate this process, let us consider a situation where we want to convey to the user the following propositions: [Wallaby hop] and [Kangaroo hop]. In the first stage, our procedure generates two RDs that can convey the proposition [Wallaby hop]: Assert[Wallaby hop] and Instantiate[Wallaby hop], where the Instantiation is performed with respect to a wallaby called Wally that is known to the user. Our procedure generates only the RD Assert[Kangaroo hop] to convey the proposition [Kangaroo hop] (an Instantiation is not generated since the user is not familiar with any particular kangaroos). In the forward reasoning stage, the inferences from these RDs are considered. If the user knows that wallabies are similar to kangaroos, the RD generated to convey the proposition [Kangaroo hop] can increase the user's belief in the proposition [Wallaby hop], and is therefore added to the *RD-list* of [Wallaby hop]. Similarly, the RDs generated to convey [Wallaby hop] are added to the *RD-list* of [Kangaroo hop]. From the above Assertions the user may also infer incorrectly that wombats hop. In this case, a proposition which negates this incorrect conclusion, i.e., [Wombat ¬hop], is assigned to $\{newp\}$. The RDs that can convey this proposition in our example are Negate[Wombat hop] and Instantiate[Wombat ¬hop], where the Instantiation is performed with respect to a wombat called Wimpy that is known to the user. These RDs in turn may yield the incorrect inferences that neither wallabies nor kangaroos hop, which contradict the intended propositions. However, since the propositions that contradict these inferences already exist in $\{oldp\}$, the process stops.

3.2.2 Constructing Minimally Sufficient Sets of RDs

In this step, we generate all the minimally sufficient sets of RDs that can convey jointly all the intended propositions. For each proposition p_i, we first determine whether RDs that were generated to convey other propositions can decrease a user's belief in p_i. Next, for each RD in *RD-list*(p_i), we determine whether it can overcome the detrimental effect of these 'negative' RDs. This step identifies combinations of RDs that cannot succeed in conveying the intended propositions. It results in the following labelling of the RDs in *RD-list*: RDs that can overcome all negative effects are labelled with the symbol [all] (the only RDs that may be labelled in this manner are Assertions and Negations); RDs that cannot convey an intended proposition by themselves are labelled with the symbol [-]; RDs that can convey an intended proposition, but cannot overcome any negative effects are labelled with [none]; and the remaining RDs are labelled with the negative RDs they can overcome.

We then use a search procedure to generate all the sets of RDs which consist of one RD from each *RD-list*. The sets of RDs that convey all the intended propositions are then stored in a list called *SUCCEED*; and the sets of RDs that fail to convey one or more propositions are stored in a list called *FAILED*, together with the proposition(s) that failed to be conveyed. Additional minimally sufficient sets of RDs are then generated from *FAILED* as follows: we select a proposition p_i that was not conveyed, and create pairs of RDs composed of the RD that failed to convey p_i and each of the other RDs in *RD-list*(p_i) that is not labelled [all] (the RDs that are labelled [all] can convey p_i by themselves, and therefore there is no need to combine them with other RDs). Each pair of RDs inherits the negative RDs that can be overcome by its parents, and may be able to overcome additional negative RDs which caused its parents to fail separately. For each pair of RDs, a new set of RDs is created by replacing the RD which failed to convey p_i with this pair of RDs. The search is then continued for each of these new sets of RDs until a minimally sufficient set of RDs is generated or failure occurs again. In this case, the process of generating pairs of RDs is repeated, and the search is resumed. If a pair of RDs fails, then it forms the basis for triplets, and so on. The search stops when the *RD-list* of a proposition which failed to be conveyed contains no RDs with which the failed RDs (or RD-tuples) can be combined.

Algorithm *Construct-sets-of-RDs*($\{p\}$, *RD-list*)
1. Initialize two lists, *SUCCEED* and *FAILED*, to empty.
2. Determine-RDs(*nil*,$\{p\}$,*nil*).
3. For each proposition $p_i \in \{p\}$
  (a) Put in *NegRDs*(p_i) all the RDs in *RD-list* that have a negative effect on p_i.
  (b) Label each RD in *RD-list*(p_i) according to the RDs in *NegRDs*(p_i) whose effect it can overcome. If there are several RDs in *NegRDs*(p_i) then all the combinations of these RDs must be considered.
4. Exhaustively generate all the combinations of RDs consisting of one RD from the *RD-list* of each proposition. Consider the combined effect of several RDs to determine whether a set of RDs conveys completely a set of propositions.
5. Append the successful combinations of RDs to *SUCCEED*, and remove any sets of RDs in *SUCCEED* that subsume other sets of RDs.
6. Append the failed combinations of RDs to *FAILED* together with the reason for the failure, i.e., the RDs that failed and the propositions that were not conveyed.
7. If *FAILED* is empty, then exit.
8. Assign to *CURRD* the first set of RDs in *FAILED*, and remove it from *FAILED*.
9. Select from *CURRD* a proposition p_i that was not conveyed, and generate successors of *CURRD* as follows:
  (a) Generate children of the RD that failed to convey p_i by combining it with other RDs in *RD-list*(p_i) that are not labelled [all].
  (b) If the failed RD has no children in p_i, then remove from *FAILED* all the sets of RDs which failed when this RD tried to convey p_i, and go to Step 7.

Table 2. Sample RDs (labelled)

Proposition	RDs and labels	
p_1: [Wallaby hop]	$\mathbf{A}(p_1)$	[all]
	$\mathbf{I}(p_1)$	[none]
	$\mathbf{A}(p_2)$	[-]
p_2: [Kangaroo hop]	$\mathbf{A}(p_2)$	[all]
	$\mathbf{A}(p_1)$	$[\mathrm{I}(\neg p_3)]$
	$\mathbf{I}(p_1)$	[-]
$\neg p_3$: [Wombat $\neg$hop]	$\mathbf{N}(p_3)$	$[\mathrm{A}(p_1)+\mathrm{I}(p_1), \mathrm{A}(p_2)]$
	$\mathbf{I}(\neg p_3)$	$[\mathrm{I}(p_1), \mathrm{A}(p_1)]$

(c) Attach to each combination of RDs the list of negative RDs it can overcome.

(d) Create sets of RDs such that in each set the failed RD is replaced with one of its children.

10. Go to Step 5.

To illustrate this process, let us reconsider the example discussed in Section 3.2.1. Table 2 contains the *RD-lists* for the propositions in this example, where each RD is labelled according to the RDs whose negative effect it can overcome. For instance, **Negate**(p_3) can overcome the combined negative effect of **Assert**(p_1) and **Instantiate**(p_1), and also the effect of **Assert**(p_2). However, it cannot overcome the combined effect of **Assert**(p_1) and **Assert**(p_2), or **Assert**(p_2) and **Instantiate**(p_1). **Instantiate**(p_1) can convey proposition p_1, but cannot overcome any negative effects. **Assert**(p_2) contributes to the belief in p_1 but cannot convey it alone.

Figure 2 contains part of the search tree generated in Step 4 of algorithm *Construct-sets-of-RDs*. Each path in this tree contains one RD from each row in Table 2. Successful paths are drawn with thick lines and are marked **S**. Failed paths are marked **F** accompanied by the propositions which were not conveyed by the RDs in these paths. An RD that increases a user's belief in more than one proposition may appear in a path more than once. The repeated instances of such an RD appear in brackets, e.g., {**Assert**(p_1)}, indicating that the RD will be mentioned only once. In the successful path to the left, **Assert**(p_1) can overcome all negative effects to convey p_1. In addition, it can overcome the negative effect of **Instantiate**($\neg p_3$) to convey p_2, and **Instantiate**($\neg p_3$) can overcome the negative effect of **Assert**(p_1) to convey $\neg p_3$.

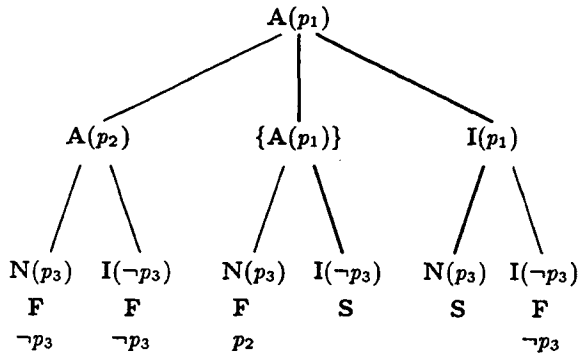

Figure 2. Partial Sample Search Tree

Table 3. Sets of RDs after the Initial Search

SUCCESSFUL	1. **Assert**(p_1) **Instantiate**($\neg p_3$)	
	2. **Assert**(p_1) **Instantiate**(p_1) **Negate**(p_3)	
FAILED	3. **Assert**(p_1) **Assert**(p_2) **Negate**(p_3)	$\neg p_3$
	4. **Assert**(p_1) **Negate**(p_3)	p_2
	5. **Instantiate**(p_1) **Assert**(p_2) **Negate**(p_3)	$\neg p_3$
	6. **Instantiate**(p_1) **Assert**(p_2) **Instantiate**($\neg p_3$)	$\neg p_3$
	7. **Instantiate**(p_1) **Negate**(p_3)	$\{p_1, p_2\}$
	8. **Instantiate**(p_1) **Instantiate**($\neg p_3$)	$\{p_1, p_2\}$
	9. **Assert**(p_2) **Negate**(p_3)	p_1
	10. **Assert**(p_2) **Instantiate**($\neg p_3$)	$\{p_1, \neg p_3\}$

In the successful path to the right, **Assert**(p_1) together with **Instantiate**(p_1) overcome the negative effect of **Negate**(p_3) to convey p_2, even though neither could do so by itself; and **Negate**(p_3) can overcome the joint effect of **Assert**(p_1) and **Instantiate**(p_1).

Table 3 contains the successful minimally sufficient sets of RDs generated by this search and the failed sets of RDs accompanied by the propositions that were not conveyed. In Step 9 of *Construct-sets-of-RDs*, the RDs that failed to convey a proposition are combined with other RDs that can increase the user's belief in this proposition. For instance, **Negate**(p_3) is combined with **Instantiate**($\neg p_3$) for all the paths where $\neg p_3$ failed to be conveyed, and the search is continued. Our procedure does not generate children from repeated RDs that failed to convey a proposition, since this would yield already existing combinations of RDs. Table 4 contains the minimally sufficient sets of RDs returned by algorithm *Construct-sets-of-RDs*. Set 5-6 is obtained by complementing Set 5 in Table 3 with the RD **Instantiate**($\neg p_3$), and also by complementing Set 6 with the RD **Negate**(p_3). Additional successful sets of RDs are generated by appending complementary RDs to the failed sets of RDs in Table 3. However, these sets subsume Set 1, 2 and 5-6, and hence are not minimally sufficient. For example, when Set 4 in Table 3 is complemented with **Instantiate**(p_1), it yields a set of RDs that is equal to Set 2. This set is removed in Step 5 of *Construct-sets-of-RDs*.

This process ensures that only minimally sufficient sets of RDs are generated, because it generates RD-tuples only from the unsuccessful RDs in the *RD-list* of a proposition, and it prunes sets of RDs that subsume other sets of RDs. In addition, this process ensures that all the minimally sufficient sets of RDs are generated, because it considers all the RD-tuples resulting from the unsuccessful RDs in the *RD-list* of a proposition.

Table 4. Minimally Sufficient Sets of RDs

1. **Assert**(p_1) **Instantiate**($\neg p_3$)
2. **Assert**(p_1) **Instantiate**(p_1) **Negate**(p_3)
5-6. **Instantiate**(p_1) **Assert**(p_2) **Negate**(p_3) **Instantiate**($\neg p_3$)

3.2.3 Determining Prerequisite Propositions

The prerequisite propositions to be conveyed depend on the user's expertise with respect to the concepts mentioned in a set of RDs, and on the context where these concepts are mentioned. The context influences both the aspects of these concepts that must be understood by the user and the extent to which these aspects must be understood.

The process of determining the relevant aspects of a concept and the required level of expertise is described in [Zukerman and McConachy, 1993a]. The relevant aspects of a concept are determined by considering the predicates of the propositions where a concept is mentioned, and the role of the concept in these propositions. For example, in order to understand the RD Assert[Marsupial has-part pouch], the user must know the aspects *type* and *structure* of a pouch, i.e., what it is and what it looks like. The extent to which a user must know the selected aspects of a concept depends on the relevance of this concept to the original propositions to be conveyed, i.e., the system demands a high level of expertise with respect to the more relevant concepts, and a lower level of expertise with respect to the less relevant ones.

After the relevant aspects and required level of expertise of each concept have been determined, WISHFUL-II applies the content selection step described in Section 2 to determine the prerequisite propositions of each concept. WISHFUL-II then merges into a single set the prerequisite propositions generated for individual concepts. This merger is executed because some prerequisite propositions of two or more concepts may be conveyed by a single RD. A special case of this happens when two or more concepts have common prerequisite propositions. For example, consider the situation depicted in Figure 3, where prerequisite information for the set of RDs $\{RD_1, RD_2\}$ is being conveyed. RD_1 requires the prerequisite propositions $\{p_1, p_2\}$, while RD_2 requires the prerequisite propositions $\{p_2, p_3, p_4\}$. If we considered separately the prerequisites of these RDs, we would generate RD_3 to convey $\{p_1, p_2\}$, and $\{RD_4, RD_5\}$ to convey $\{p_2, p_3, p_4\}$. This would result in a total of three RDs. However, by considering jointly all the prerequisite propositions of $\{RD_1, RD_2\}$, we will require two RDs only, namely $\{RD_3, RD_5\}$.

3.2.4 Evoking the Concepts in a Set of RDs

RDs that convey referring information differ from RDs that convey prerequisite propositions in that the former identifies a concept by means of information known to the user, while the latter conveys information that the user does not know about a concept. Further, the process of generating referring information has the flexibility of selecting the propositions that can identify a concept uniquely, while the propositions

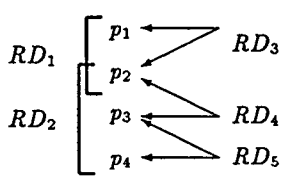

Figure 3. Prerequisite Propositions of a Set of RDs

Table 5. Sample Referring Expressions

Concept	Lexical Item	Complementing Information
Like-Terms	"Like Terms"	$Comp_1$: Algebraic Terms with one variable $Comp_2$: expressions such as $2(3x+4x)$
Algebraic-Terms	"Algebraic Terms"	$Comp_3$: both Like Terms and Unlike Terms $Comp_4$: expressions such as $2(3x+4y)$ and $5(2z+3z)$

that convey prerequisite information are dictated by the context and by the user's expertise.

In order to generate referring expressions for the concepts mentioned in a set of RDs, we propose for each concept a list of candidate lexical items that can be used to refer to it. If there is a lexical item that identifies each concept uniquely and is known to the user, the evocation process is finished. However, if there are concepts that are not identified uniquely by any of their candidate lexical items, then these lexical items are complemented with additional RDs that help them identify the intended concepts. This task is performed by iteratively selecting propositions that identify an intended concept until this concept is singled out, and generating RDs that convey these propositions. This algorithm differs from the procedure described in [Dale, 1990] in that we generate several alternative sets of complementing RDs in order to avoid dead-end situations where the only identifying information that is generated for a set of concepts is circular. The evocation process then selects the most concise non-circular combination of referring expressions that identifies all the concepts in a set of RDs. For example, Table 5 illustrates candidate referring expressions generated for the concepts Like-Terms and Algebraic-Terms. Each referring expression is composed of a lexical item and a complement[3]. The non-circular alternatives in this example contain the complements $\{Comp_1, Comp_4\}$, $\{Comp_2, Comp_3\}$ and $\{Comp_2, Comp_4\}$.

3.3 Two Optimization Criteria

As indicated in Section 3.1, the optimization criterion determines the manner in which the nodes in *OPEN* are ranked and pruned. In our implementation we have tried two optimization criteria: (1) conciseness and (2) depth.

3.3.1 Optimizing the Depth of the Generated RDs

When optimizing the depth of a set of RDs, the nodes in *OPEN* are pruned according to the following rule:

IF Depth(n_i) = Depth(n_j) AND
{ Prerequisites-of(n_i) ⊃ Prerequisites-of(n_j) OR
{ Prerequisites-of(n_i) = Prerequisites-of(n_j) AND
{ |Referring-exp-of(n_i)| > |Referring-exp-of(n_j)| OR
{ |Referring-exp-of(n_i)| = |Referring-exp-of(n_j)| AND
Total-Weight(n_i) ≥ Total-Weight(n_j) } } } }
THEN remove n_i.

[3] A referring expression may contain a null lexical item, i.e., complementing information only. However, at present this option is not generated by WISHFUL-II.

Table 6. Prerequisite Propositions and Total Weight of Minimally Sufficient Sets of RDs

Set of RDs	Total Weight	Prerequisite Propositions
1. Assert(p_1) Instantiate($\neg p_3$)	2	p_{11}, p_{12}, p_{13}
2. Assert(p_1) Instantiate(p_1) Negate(p_3)	2.5	p_{11}, p_{12}, p_{13}, p_{31}
5-6. Instantiate(p_1) Assert(p_2) Negate(p_3)Instantiate($\neg p_3$)	3.5	p_{21}, p_{31}

The *weight* of a node reflects the number of RDs in this node and their type. The *total weight* of a node is the sum of the weights of the nodes in the path from the root of the search tree to this node. All the RDs have a weight of 1, except an Instantiation of a proposition p that accompanies an Assertion of p or a Negation of $\neg p$. Such an Instantiation has a weight of $\frac{1}{2}$, because it does not contain new information, rather it is a continuation of the idea presented in the Assertion or the Negation. For example, the Instantiation in Set 1 in Table 6 has a weight of 1, because the instantiated proposition is different from the asserted proposition. In contrast, in Set 2, the weight of the Instantiation is $\frac{1}{2}$ because it instantiates the asserted proposition.

The above rule is also applied after Step 4 of algorithm *Expand-sets-of-RDs* to prune the list of minimally sufficient sets of RDs (Section 3.2). It removes a node if its prerequisites subsume those of another node. It considers the number of referring expressions of a node only when the same prerequisite propositions are required by two nodes, and considers the total weight of a node only when two nodes have the same prerequisite propositions and the same number of referring expressions. This rule compares only nodes at the same depth, because even if the prerequisites of a node at level i subsume the prerequisites of a node at level $i+1$, the node at level i may lead to discourse that has depth $i+1$, while the node at level $i+1$ can lead to discourse of depth $i+2$ at best.

To illustrate the pruning process let us reconsider the minimally sufficient sets of RDs in Table 4, assuming that the prerequisite propositions required by these sets are as shown in Table 6[4]. Here, the pruning rule removes Set 2, since its prerequisite propositions subsume those of Set 1.

The nodes remaining in *OPEN* are ordered as follows:

1. In increasing order of their depth, so that we expand the more shallow nodes first during the optimization process.
2. In increasing order of the number of prerequisite propositions they require, so that the nodes that contain the sets of RDs with the fewest prerequisites are preferred among the nodes at the same level.
3. In increasing order of the number of referring expressions they have, so that the nodes with the fewest referring expressions are preferred among the nodes with the same number of prerequisite propositions.
4. In increasing order of their total weight, so that the most concise set of RDs is preferred when all else is equal.

[4] The first coefficient of each prerequisite proposition indicates the RD for which it is required, e.g., p_{12} is a prerequisite of Assert(p_1).

To illustrate this process let us consider the minimally sufficient sets of RDs that remain after pruning, namely Set 1 and Set 5-6, and assign them to nodes n_1 and n_{5-6} respectively. Since Set 5-6 has the fewest prerequisite propositions, n_{5-6} will precede n_1 in *OPEN*, and will be the next node to be expanded by algorithm *Optimize-RDs* (Section 3.1). If upon expansion of n_{5-6} we find that there is a minimally sufficient set of RDs that conveys propositions $\{p_{21}, p_{31}\}$ and requires no prerequisite information, then the node which contains this set of RDs is a goal node, and the search is finished.

3.3.2 *Optimizing the Number of Generated RDs*

When optimizing the total number of RDs to be presented, the following rule is used to prune the nodes in *OPEN*:
IF Total-Weight(n_i) $\geq$ Total-Weight(n_j) AND
Prerequisites-of(n_i) $\supseteq$ Prerequisites-of(n_j)
THEN remove n_i.

As in depth optimization, this rule is also applied after Step 4 of algorithm *Expand-sets-of-RDs* to prune the list of minimally sufficient sets of RDs.

The nodes remaining in *OPEN* are sorted in increasing order of their total weight.

To illustrate this process let us consider once more the minimally sufficient sets of RDs in Table 6. Since the prerequisite propositions of Set 2 subsume those of Set 1, and the total weight of Set 2 is higher than that of Set 1, Set 2 is removed in the pruning stage. The ordering of the remaining nodes in *OPEN* is different from the ordering obtained for the depth optimization, i.e., n_1 precedes n_{5-6} in *OPEN*, since the total weight of Set 1 is less than the total weight of Set 5-6.

4 Results

WISHFUL-II was implemented using Common Lisp on a SPARCstation 2 and on a PC-486. The system takes less than 4 seconds of CPU time to produce English output, and the optimization process alone takes less than 2 seconds for discourse of up to 10 RDs. Table 1 in Section 1 and Table 7 (adapted from an example in [Moore and Swartout, 1989]) illustrate the output generated by WISHFUL-II for the two optimization criteria we have implemented, viz conciseness and depth. Appendix A contains examples of the output produced by WISHFUL-II when the same discourse is generated for the concise and the shallow optimization criteria.

Our mechanism can be used as a tool for evaluating different discourse optimization criteria, where the only requirement for implementing a new criterion is the modification of the pruning and ranking rules described in Section 3.3. When WISHFUL-II was tried with the two optimization criteria described in this paper, it often generated the same discourse with both criteria, i.e., the most concise discourse was also the shallowest. However, the two optimization criteria produced different discourse when the most concise discourse mentioned one or more concepts that were not known to the user and therefore had to be explained, while the shallowest discourse avoided these explanations by presenting a larger

Table 7. Sample Concise and Shallow 'Lisp' Discourse

Concise Discourse	Shallow Discourse
setq is like setf. However, the first argument of setq is not a generalized variable, which is an expression that references a storage location. The first argument of setq is a simple variable.	setq takes two arguments. The first argument of setq is a simple variable. The second argument of setq is a value. The objective of setq is to assign the value to the variable. For example, (setq x '(a b)) results in x-->'(a b).

number of RDs which mentioned different concepts. In particular, the concise discourse was characterized by the presence of Similes that required some in-depth clarification of a non-source concept[5], while the shallow discourse was characterized by the presence of a Description composed of a list of Assertions possibly accompanied by Instantiations.

5 Conclusion

In this paper, we have cast discourse planning as an optimization process which generates discourse that satisfies a specific optimization criterion. We have described a weak search procedure that implements this process while taking into consideration the following factors: (1) a user's inferences from proposed RDs, (2) the prerequisite information required by the user to understand the concepts mentioned in a set of RDs, and (3) the referring expressions required to enable the user to identify these concepts. Two optimization criteria have been considered, viz conciseness and depth. The system which implements these ideas has been used to generate descriptive discourse in various technical domains.

REFERENCES

A. Cawsey (1990), Generating Explanatory Discourse. In: R. Dale, C. Mellish and M. Zock, eds., *Current Research in Natural Language Generation* (Academic Press), pp. 75-102.

R. Dale (1990), Generating Recipes: An Overview of Epicure. In: R. Dale, C. Mellish and M. Zock, eds., *Current Research in Natural Language Generation* (Academic Press), pp. 229-255.

M. Elhadad (1992), FUG: The Universal Unifier User Manual Version 5.0, Technical Report, Columbia University, New York, New York.

M.R. Garey and D.S. Johnson (1979). *Computers and Intractability, A Guide to the Theory of NP-Completeness* (W.H. Freeman and Company, San Francisco).

K. McKeown (1985), Discourse Strategies for Generating Natural Language Text, *Artificial Intelligence* 27(1), pp. 1-41.

J.D. Moore and W.R. Swartout (1989), A Reactive Approach to Explanation. In: *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Michigan, pp. 1504-1510.

[5] A source concept in a Simile is a concept from which the intended information is drawn, e.g., kangaroo in Table 1 and setf in Table 7 are source concepts; pouch in Table 1 and generalized-variable in Table 7 are non-source concepts.

N. Nilsson (1980), *Principles of Artificial Intelligence* (Tioga Publishing Company, Palo Alto, California).

C.L. Paris (1988), Tailoring Object Descriptions to a User's Level of Expertise, *Computational Linguistics* 14(3), pp. 64-78.

J. Weiner (1980), Blah, A System Which Explains Its Reasoning, *Artificial Intelligence* 15, pp. 19-48.

I. Zukerman and R.S. McConachy (1993a), Generating Concise Discourse that Addresses a User's Inferences. In: *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Chambery, France, pp.1202-1207.

I. Zukerman and R.S. McConachy (1993b), An Optimizing Method for Structuring Inferentially Linked Discourse. In: *Proceedings of the National Conference on Artificial Intelligence*, Washington, D.C., pp. 202-207.

Appendix A: Sample Output

Table 8. 'Racing car' Discourse

An indycar is an American racing car. It has a very powerful engine, wide tires, huge brakes and big wings to make lots of downforce. Lots of downforce helps it go around corners quickly, however lots of downforce does not help it go straight quickly. A formula 1 car is like an indycar, however a formula 1 car is a European racing car.

Table 9. 'DOS and UNIX' Discourse

DOS is an operating system. It has a command line interface, which is an interface where you type commands at a text prompt, e.g., mkdir, ls. It is a single user operating system and it does not allow multitasking, which is doing more than one job at a time. UNIX is like DOS, however it is a multiuser operating system and does allow multitasking. Some UNIX commands are the same as DOS, e.g., mkdir, however some are different, e.g., pwd. DOS runs on PC compatibles. In addition to PC compatibles UNIX runs on workstations.

Table 10. 'Document layout' Discourse

TeX is a layout language for documents. A layout language allows you to control the appearance of of your document, e.g., text size and placement. It uses embedded commands, which are commands placed within the document, e.g., \pageno. These commands are executed after your document is edited. Troff is like TeX, however it has different commands, e.g., .BP. A wordprocessor also allows you to control the appearance of your document, however it is not a layout language. Wordprocessors also use embedded commands, however they are not executed after your document is edited, they are executed while your document is edited. WordPerfect is a wordprocessor.