Computational Intelligence, Volume 000, Number 000, 0000

Experiences with Planning for Natural Language Generation

Alexander Koller
Cluster of Excellence, Saarland University, Saarbrücken, Germany

Ronald P. A. Petrick
School of Informatics, University of Edinburgh, Edinburgh, UK

Natural language generation (NLG) is a major subfield of computational linguistics with a long tradition as an application area of automated planning systems. While things were relatively quiet with the planning approach to NLG for a while, several recent publications have sparked a renewed interest in this area. In this paper, we investigate the extent to which these new NLG approaches profit from the advances in planner expressiveness and efficiency. Our findings are mixed. While modern planners can readily handle the search problems that arise in our NLG experiments, their overall runtime is often dominated by the grounding step they perform as preprocessing. Furthermore, small changes in the structure of a domain can significantly shift the balance between search and preprocessing. Overall, our experiments show that the off-the-shelf planners we tested are unusably slow for nontrivial NLG problem instances. As a result, we offer our domains and experiences as challenges for the planning community.

Key words: natural language generation, planning

1. INTRODUCTION

Natural language generation (NLG; Reiter and Dale 2000) is one of the major subfields of natural language processing, concerned with computing natural language sentences or texts that convey a given piece of information to an audience. While the output of a generation task can take many forms, including written text, synthesised speech, or embodied multimodal presentations, the underlying NLG problem in each case can be modelled as a problem of achieving a (communicative) goal by successively applying a set of (communicative) actions. This view of NLG as goal-directed action has clear parallels to *automated planning*, which seeks to find general techniques for efficiently solving the action sequencing problem.

Treating generation as planning has a long history in NLG, ranging from the initial attempts of the field to utilise early planning approaches (Perrault and Allen 1980; Appelt 1985; Hovy 1988; Young and Moore 1994), to a recent surge of research (Steedman and Petrick 2007; Koller and Stone 2007; Brenner and Kruijff-Korbayová 2008; Benotti 2008) seeking to capitalise on the improvements modern planners offer in terms of efficiency and expressiveness. This paper attempts to assess the usefulness of current planning techniques to NLG by investigating some representative generation problems, and by evaluating whether automated planning has advanced to the point that it can provide solutions to such NLG applications—applications that are not currently being investigated by mainstream planning research.

To answer this question, we proceed in two ways. First, we present two generation problems that have recently been cast as planning problems: the sentence generation task and the GIVE task. In the sentence generation task, we concentrate on generating a single sentence that expresses a given meaning. In this case, a plan encodes the necessary sentence with the actions in the plan corresponding to the utterance of individual words (Koller and Stone 2007). In the GIVE domain ("Generating Instructions in Virtual Environments"), we describe a new shared task that was recently posed as a challenge for the NLG community (Byron et al. 2009). GIVE uses planning as part

[1] Address correspondence to koller@mmci.uni-saarland.de or rpetrick@inf.ed.ac.uk.

of a larger NLG system for generating natural-language instructions that guide a human user in performing a given task in a virtual environment.

Second, we evaluate the performance of several off-the-shelf planners on the planning domains into which these two generation problems translate. Among the planners we test, we explore the efficiency of FF (Hoffmann and Nebel 2001)—a planner that has arguably had the greatest impact on recent approaches to deterministic planning—and some of its descendants, such as SGPLAN (Hsu et al. 2006). All of the planners we test are freely available, support an expressive subset of the Planning Domain Definition Language (PDDL; McDermott et al. 1998), and have been successful on both standard planning benchmarks and the problems of the International Planning Competition (IPC).[1] Using these planners—together with an ad-hoc Java implementation of GraphPlan (Blum and Furst 1997) serving as a baseline for certain experiments—we perform a series of tests on a range of problem instances in our NLG domains.

Overall, our findings are mixed. On the one hand, we demonstrate that some planners can readily handle the *search* problems that arise in our testing domains on realistic inputs, which is promising given the challenging nature of these tasks (e.g., the sentence generation task is NP-complete; see Koller and Striegnitz 2002). On the other hand, these same planners often spend tremendous amounts of time on *preprocessing* to analyse the problem domain in support of the search. On many of our problem instances, the preprocessing time overshadows the search time. (For instance, FF spends 90% of its runtime in the sentence generation domain on preprocessing.) Furthermore, small changes in the structure of a planning domain can dramatically shift the balance between preprocessing and search. As a consequence, we are forced to conclude that the off-the-shelf planners we investigated are generally too slow to be useful in real NLG applications. It is also our hope, however, that these results will spark an interest to improve the quality of planner implementations—especially in the area of preprocessing techniques—and to this end we offer our domains and experiences as challenges for the planning community.

The remainder of this paper is structured as follows. In Section 2, we introduce the idea of NLG as planning and briefly review the relevant literature. In Section 3, we describe a set of planning problems associated with two NLG tasks: sentence planning and situated instruction generation. In Section 4, we report on our experiments with these planning problems. In Section 5 we discuss our results and overall experiences, and conclude in Section 6.

2. NLG AS PLANNING

The task of generating natural language from semantic representations (NLG) is typically split into two parts: the *discourse planning* task, which selects the information to be conveyed and structures it into sentence-sized chunks, and the *sentence generation* task, which then translates each of these chunks into natural language sentences. The sentence generation task is often divided into two parts of its own—the *sentence planning* task, which enriches the input by, e.g., determining object references and selecting some lexical material, and the *surface realization* task, which maps the enriched meaning representation into a sentence using a grammar. The chain of domain planning, sentence planning, and surface realization is sometimes called the "NLG pipeline" (Reiter and Dale 2000).

Viewing generation as a planning problem has a long tradition in the NLG literature. Perrault and Allen (1980) presented an approach to discourse planning in which the planning operators represented individual speech acts such as "request" and "inform". This idea was later expanded, e.g., by Young and Moore (1994). On the other hand, researchers such as Appelt (1985) and Hovy (1988) used techniques from hierarchical planning to expand a high-level plan consisting of speech acts into more detailed specifications of individual sentences. Although these systems covered some aspects of sentence planning, they also used very expressive logics designed to reason about beliefs and intentions, in order to represent the planning state and the planning operators. Most of these systems also used ad-hoc planning algorithms with rather naive search strategies, which did not scale

[1] See `http://ipc.icaps-conference.org/` for information about past editions of the IPC. Also see (Hoffmann and Edelkamp 2005) for a good overview of the deterministic track of the 2004 competition.

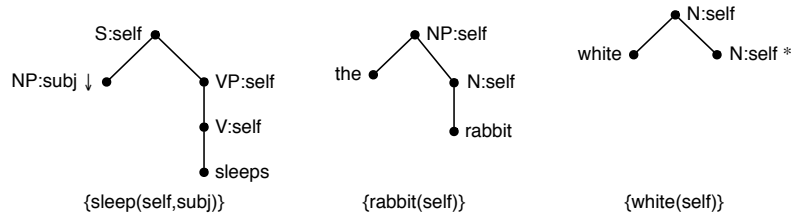


Figure 1: An example grammar in the sentence generation domain.

well to realistic inputs. As a consequence, the NLG-as-planning approach was mostly marginalized throughout the 1990s.

More recently, there has been a string of publications by various authors with a renewed interest in the generation-as-planning approach, motivated by the ongoing development of increasingly more efficient and expressive planners. For instance, Koller and Stone (2007) propose an approach to sentence generation (i.e., the sentence planning and surface realization modules of the pipeline) as planning—an approach we explore in more detail below (Section 3.1). Steedman and Petrick (2007) revisit the analysis of indirect speech acts with modern planning technology, viewing the problem as an instance of planning with incomplete information and sensing actions. In addition, Benotti (2008) uses planning to explain the accommodation of presuppositions, and Brenner and Kruijff-Korbayová (2008) use multi-agent planning to model the joint problem solving behaviour of agents in a situated dialogue. While these approaches focus on different issues compared to the 1980's NLG-as-planning literature, they all apply *existing*, well-understood planning approaches to linguistic problems, in order to utilise the rich set of modelling tools provided by modern planners, and in the hope that such planners can efficiently solve the hard search problems that arise in NLG (Koller and Striegnitz 2002). This paper aims to investigate whether existing planners achieve this latter goal.

3. TWO NLG TASKS

We begin by considering two specific NLG problems: sentence generation in the sense of Koller and Stone (2007), and the generation of instructions in virtual environments (Byron et al. 2009). In each case, we introduce the task and show by example how it can be viewed as a planning problem.

3.1. Sentence generation as planning

One way of modelling the sentence generation problem is to assume a lexicalized grammar in which each lexicon entry specifies how it can be combined grammatically with the other lexicon entries, what piece of meaning it expresses, and what the pragmatic conditions on using it are. Sentence generation can then be seen as constructing a grammatical derivation that is syntactically complete, respects the semantic and pragmatic conditions, and achieves all the *communicative goals*.

An example of such a lexicalized grammar is the tree-adjoining grammar (TAG; Joshi and Schabes 1997) shown in Figure 1. This grammar consists of *elementary trees* (i.e., the disjoint trees in the figure), each of which contributes certain *semantic content*. For instance, say that a knowledge base contains the individuals e, r_1 and r_2, and the facts that r_1 and r_2 are rabbits, r_1 is white and r_2 is brown, and e is an event in which r_1 sleeps. We could then construct a sentence expressing the information $\{\mathsf{sleep}(e, r_1)\}$ by combining instances of the elementary trees (in which the *semantic roles*, such as self and subj, have been substituted by constants from the knowledge base) into a TAG derivation as shown in Figure 2. In the figure, the dashed arrow indicates TAG's *substitution* operation, which "plugs" an elementary into the leaf of another tree; the dotted arrows stands for *adjunction*, which splices an elementary tree into an internal node. We can then read the sentence "The white rabbit sleeps" from the derivation. Note that the sentence "The rabbit sleeps" would not have been an appropriate result, because "the rabbit" could refer to either r_1 or r_2. Thus, r_2 remains as a *distractor*, i.e., an incorrect possible interpretation of the phrase.

This perspective on sentence generation also has the advantage of solving the sentence planning and surface realization problems simultaneously, which is particularly useful in cases where these

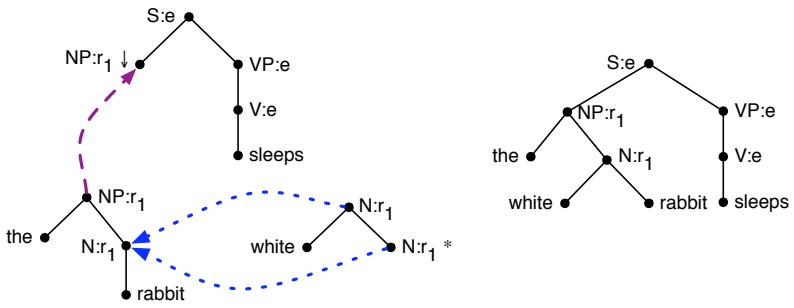


FIGURE 2: Derivation of "The white rabbit sleeps."

```
(:action add-sleeps
   :parameters (?u - node  ?xself - individual  ?xsubj - individual)
   :precondition
       (and (subst S ?u)  (referent ?u ?xself)  (sleep ?xself ?xsubj))
   :effect
       (and (not (subst S ?u))  (expressed sleep ?xself ?xsubj)
            (subst NP (subj ?u))  (referent (subj ?u) ?xsubj)
            (forall (?y - individual)
                (when (not (= ?y ?xself)) (distractor (subj ?u) ?y)))))

(:action add-rabbit
   :parameters (?u - node  ?xself - individual)
   :precondition
       (and (subst NP ?u)  (referent ?u ?xself)  (rabbit ?xself))
   :effect
       (and (not (subst NP ?u))  (canadjoin N ?u)
            (forall (?y - individual)
                (when (not (rabbit ?y)) (not (distractor ?u ?y))))))

(:action add-white
   :parameters (?u - node  ?xself - individual)
   :precondition
       (and (canadjoin N ?u)  (referent ?u ?xself)  (rabbit ?xself))
   :effect
       (forall (?y - individual)
           (when (not (white ?y)) (not (distractor ?u ?y)))))
```

FIGURE 3: PDDL actions for generating the sentence "The white rabbit sleeps."

two problems interact. For instance, the generation of referring expressions (REs) is usually seen as a sentence planning task, however, syntactic information about individual words in available when the REs are generated (see, e.g., Stone and Webber 1998). (In the example, we require the referring expression "the white rabbit" to be resolved uniquely to r_1 by the hearer, in addition to the requirement that the derivation be grammatically correct.)

However, the problem of deciding whether a given communicative goal can be achieved with a given grammar is NP-complete (Koller and Striegnitz 2002): a naive search algorithm that computes a derivation top-down takes exponential time and is clearly infeasible to use in practice. In order to circumvent this combinatorial explosion, the seminal SPUD system (Stone et al. 2003), which first established the idea of integrated TAG-based sentence generation, used a greedy, but incomplete, search algorithm. To better control the search, Koller and Stone (2007) recently proposed an alternative approach which converts the sentence generation problem into a planning problem, and solves the transformed search problem using a planner (Koller and Stone 2007).[2] The resulting planning

[2]See `http://code.google.com/p/crisp-nlg/` for the CRISP system, which implements this conversion.

problem in this case assumes an initial state containing an atom $\mathsf{subst}(S, \mathsf{root})$, encoding the fact that a sentence (S) must be generated starting at the node named root in the TAG derivation tree, and a second atom $\mathsf{referent}(\mathsf{root}, e)$ which encodes the fact that the entire sentence describes the (event) individual e. The elementary trees in the TAG derivation are encoded as individual planning operators.

Figure 3 shows the transformed planning operators needed to generate the above example sentence, "The white rabbit sleeps." Here the action instance $\mathsf{add\text{-}sleeps}(\mathsf{root}, e, r_1)$ replaces the atom $\mathsf{subst}(S, \mathsf{root})$ with the atom $\mathsf{subst}(NP, \mathsf{subj}(\mathsf{root}))$. In an abuse of PDDL syntax, we write $\mathsf{subj}(\mathsf{root})$ as a shorthand for a fresh individual name.[3] At the same time, the operator records that the semantic information $\mathsf{sleep}(e, r_1)$ has now been expressed, and introduces all individuals except r_1 as distractors for the new RE at $\mathsf{subj}(\mathsf{root})$. These distractors can then be removed by subsequent applications of the other two operators. Eventually we reach a goal state, which is characterized by goals including $\forall x \forall y. \neg \mathsf{subst}(x, y)$, $\forall x \forall y. \neg \mathsf{distractor}(x, y)$, and $\mathsf{expressed}(\mathsf{sleep}, e, r_1)$. For instance, the following plan correctly performs the necessary derivation:

(1) $\mathsf{add\text{-}sleeps}(\mathsf{root}, e, r_1)$,
(2) $\mathsf{add\text{-}rabbit}(\mathsf{subj}(\mathsf{root}), r_1)$,
(3) $\mathsf{add\text{-}white}(\mathsf{subj}(\mathsf{root}), r_1)$.

The grammatical derivation in Figure 2, and therefore the generated sentence "the white rabbit sleeps," can be systematically reconstructed from this plan. Thus, we can solve the sentence generation problem via the detour through planning and bring current search heuristics for planning to bear on generation.

3.2. Planning in instruction giving

In the second application of planning in NLG, we consider the recent GIVE Challenge ("Generating Instructions in Virtual Environments"; Byron et al. 2009). The object of this shared task is to build an NLG system which produces natural language instructions which guide a human user in performing a task in a virtual environment. From an NLG perspective, GIVE makes for an interesting challenge since it is a theory-neutral task that exercises all components of an NLG system, and emphasizes the study of communication in a (simulated) physical environment. Furthermore, because the client displaying the 3D environment to the user can be physically separated from the NLG system (provided they are connected over a network), such systems can be cheaply evaluated over the Internet. This provides a potential solution to the long-standing problem of evaluating NLG systems. The first instalment of GIVE (GIVE-1) evaluated five NLG systems on the performance of 1143 users, making it the largest ever NLG evaluation effort to date in terms of human users.

Planning plays a central role in the GIVE task. For instance, consider the example GIVE world shown in Figure 4. In this world, the user's task is to pick up a trophy in the top left room. The trophy is hidden in a safe behind a picture; to access it, the user must push certain buttons in order to move the picture out of the way, open the safe, and open doors. The user must navigate the world and perform these actions in the 3D client; the NLG system must instruct the user on how to do this. To simplify both the planning and the NLG task, the world is discretised into a set of tiles of equal size. The user can turn by 90 degree steps in either direction, and can move from the centre of one tile to the centre of the next tile, provided the path between two tiles is not blocked. Figure 5 shows the encoding of some of the available GIVE domain actions in PDDL syntax. In the example, the shortest plan to solve the task consists of 108 action steps, with the first few steps as follows:

(1) $\mathsf{turn\text{-}left}(\mathsf{north}, \mathsf{west})$,
(2) $\mathsf{move}(\mathsf{pos\_5\_2}, \mathsf{pos\_4\_2}, \mathsf{west})$,
(3) $\mathsf{manipulate\text{-}button\text{-}off\text{-}on}(\mathsf{b1}, \mathsf{pos\_5\_2})$,
(4) $\mathsf{turn\text{-}right}(\mathsf{west}, \mathsf{north})$.

[3] These terms are not valid in ordinary PDDL but can be eliminated by estimating an upper bound n for the plan length, making n copies of each action, ensuring that copy i can only be applied in step i, and replacing the term $\mathsf{subj}(u)$ in an action copy by the constant subj_i. The terms S, NP, and N are constants.

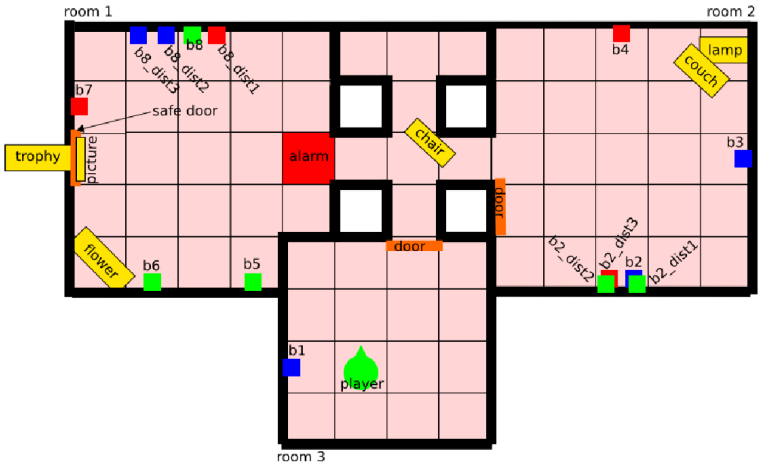


FIGURE 4: Map of an example GIVE world.

```
(:action move
   :parameters (?from - position  ?to - position  ?ori - orientation)
   :precondition
       (and (player-position ?from)  (player-orientation ?ori)
            (adjacent ?from ?to ?ori)  (not (alarmed ?to)))
   :effect
       (and (not (player-position ?from))  (player-position ?to)))

(:action turn-left
   :parameters (?ori - orientation  ?newOri - orientation)
   :precondition
       (and (player-orientation ?ori)  (next-orientation ?ori ?newOri))
   :effect
       (and (not (player-orientation ?ori))  (player-orientation ?newOri)))

(:action turn-right
   :parameters (?ori - orientation  ?newOri - orientation)
   :precondition
       (and (player-orientation ?ori)  (next-orientation ?newOri ?ori))
   :effect
       (and (not (player-orientation ?ori))  (player-orientation ?newOri)))

(:action manipulate-button-off-on
   :parameters (?b - button  ?pos - position  ?alarm - position)
   :precondition
       (and (state ?b off)  (player-position ?pos)  (position ?b ?pos)
            (controls-alarm ?b ?alarm))
   :effect
       (and (not (state ?b off))  (not (alarmed ?alarm))  (state ?b on)))
```

FIGURE 5: Simplified PDDL actions for the GIVE domain.

Our description of GIVE as a planning problem makes it very similar to the classic Gridworld problem (see, e.g., Tovey and Koenig 2000 or the 1998 edition of the IPC[4]), which also involves route finding through a two-dimensional world map with discrete positions. As in Gridworld, the domain also requires the execution of certain object-manipulation actions (e.g., finding keys and opening

[4]See `ftp://ftp.cs.yale.edu/pub/mcdermott/aipscomp-results.html`.

locks in Gridworld, or pushing the correct buttons to open doors and the safe in GIVE). However, the worlds we consider in GIVE tend to be much bigger than the Gridworld instances used in the 1998 planning competition, with more complex room shapes and more object types in the world.

To be successful in GIVE, an NLG system must be able to compute plans of the form described above. At a minimum, a discourse planner will call a domain planner in order to determine the content of the instructions that should be presented to the user. This relatively loose integration of NLG system and planner is the state of the art of the systems that participated in GIVE-1. However, it is generally desirable to integrate the planner and the generation system more closely than this. For instance, consider an NLG system that wants to generate the instruction sequence "walk to the centre of the room; turn right; now press the green button in front of you". Experiments with human instruction givers (Stoia et al. 2008) show that this is a pattern that they use frequently: the instruction follower is made to walk to a certain point in the world where the instruction giver can then use a referring expression ("the green button") that is easy for the follower to interpret. An NLG system must therefore integrate discourse planning and planning in the domain of the world map closely. On the one hand, the structure of the discourse is determined by the needs of the NLG system rather than the domain plan; on the other hand, the discourse planner must be aware of the way in which the instruction "turn right" is likely to change the visibility of objects. Even if an NLG system doesn't implement the generation of such discourse as planning, it must still solve a problem that subsumes the domain planning problem. For these reasons, we consider the GIVE domain planning problem as a natural part of a GIVE NLG system.

4. EXPERIMENTS

We now return to the original question of the paper: is planning technology ready for realistic applications in natural language generation? To investigate this question we consider two sets of experiments, designed to evaluate the performance of several planners on the NLG planning domains from the previous section. Starting with the CRISP domain, we first present a scenario which focuses on the generation of referring expressions with a tiny grammar (Section 4.1). We then look at a setting in which CRISP is used for surface realization with the XTAG Grammar (XTAG Research Group 2001), a large-scale TAG grammar for English (Section 4.2). In the second set of experiments we investigate the GIVE domain. We begin with a domain that is similar to the classic Gridworld (Section 4.3), and then add extra grid cells to the world that are not necessary to complete the task (Section 4.4). We also investigate the role that goal ordering plays in these problems. These experiments are configured in a way that lets us explore the scalability of a planner's search and preprocessing capabilities, and illustrate what we perceive to be one of the main limitations of current off-the-shelf planners for our applications: they often spend a long time computing ground instances, even when most of these instances are not required during plan search.

4.1. Experiment 1: Sentence generation (referring expressions)

For the first experiment on sentence generation, we exercise the ability of the CRISP system described in Section 3.1 to generate referring expressions. This problem is usually handled by the sentence planner if sentence planning and surface realization are separated; here it happens as part of the overall generation process.

We consider a series of sentence generation problems which require the planner to compute a plan representing the sentence "Mary likes the $\mathrm{Adj}_1 \ldots \mathrm{Adj}_n$ rabbit." Each problem instance assumes a target referent r, which is a rabbit, and a certain number m of further rabbits $r_1, \ldots, r_m$ that are distinguished by properties $P_1, \ldots, P_n$ with $n \leqslant m$. The problem instance is set up such that r has all properties except for P_i in common with each r_i for $1 \leqslant i \leqslant n$, and $r_{n+1}, \ldots, r_m$ have none of the properties P_i. That is, all n properties are required to describe r uniquely. The n properties are realized as n different adjectives, in any order. This setup allows us to vary the plan length (a plan with n properties will have length $n+4$) and the universe size (the universe will contain $m+1$ rabbit individuals in addition to the individuals used to encode the grammar, which have different types).

We converted these generation problem instances into planning problem instances as described in Section 3, and then ran several different planners on them. We used three off-the-shelf planners:

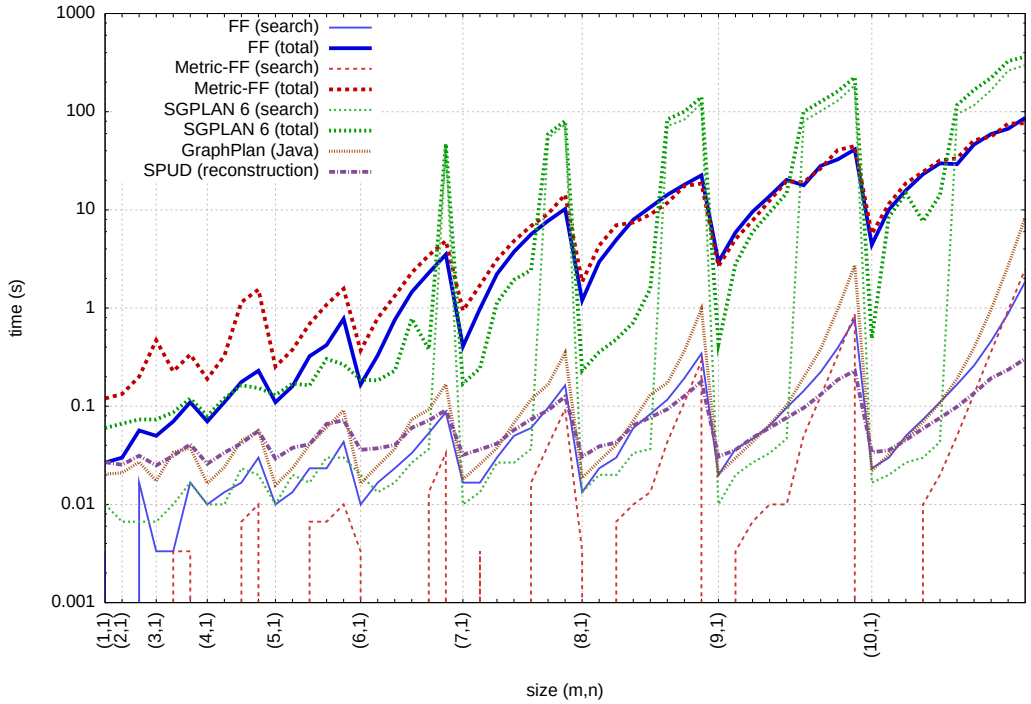


FIGURE 6: Results for the sentence generation domain. The horizontal axis represents parameters (m, n) from $(1, 1)$ to $(10, 10)$ in lexicographical order. The vertical axis is the runtime in milliseconds.

FF 2.3 (Hoffmann and Nebel 2001), Metric-FF (Hoffmann 2002), and SGPLAN 6 (Hsu et al. 2006); all of these were highly successful at the recent IPC competitions, and unlike many other IPC participants, support a fragment of PDDL with quantified and conditional effects, which is necessary in our domain. In addition, we used an ad-hoc implementation of GraphPlan (Blum and Furst 1997) written in Java; unlike the three off-the-shelf planners, it only computes instances of literals and operators as they are needed in the course of the plan search, instead of computing all ground instances in a separate preprocessing step. Finally, we reimplemented the incomplete greedy search algorithm used in the SPUD system (Stone et al. 2003) in Java.

The results of this experiment are shown in the graph in Figure 6.[5] The input parameters (m, n) are plotted in lexicographic order on the horizontal axis, and the runtime is shown in seconds on the vertical axis, on a logarithmic scale. These results reveal a number of interesting insights. First, the search times of FF and Metric-FF (shown as thinner lines) significantly outperform SGPLAN's search in this domain—on the largest instances, by a factor of over 100.[6] Second, FF and Metric-FF perform very similarly to each other, and their search times are almost the same as those of the SPUD algorithm, which is impressive because they are complete search algorithms, whereas SPUD's greedy algorithm is not.

Finally, it is striking that for all three off-the-shelf planners, the search only accounts for a tiny fraction of the total runtime; in each case, the preprocessing times are higher than the search times

[5] All runtimes in Sections 4.1 and 4.2 were measured on a single core of an AMD Opteron 8220 CPU running at 2.8 GHz, under Linux. FF 2.3 and Metric-FF were recompiled as 64-bit binaries and run with a memory limit of 32 GB. Java programs were executed under Java 1.6.0_13 in 64-bit mode and were allowed to "warm up", i.e., the JVM was given the opportunity to just-in-time compile the relevant bytecode by running the planner three times and discarding the runtimes before taking the actual measurements. All runtimes are averaged over three runs of the planners.

[6] For FF and Metric-FF, we report the "searching" and "total" times reported by the planners. For SGPLAN, we report the "total" time and the difference between the "total" and "parsing" times.

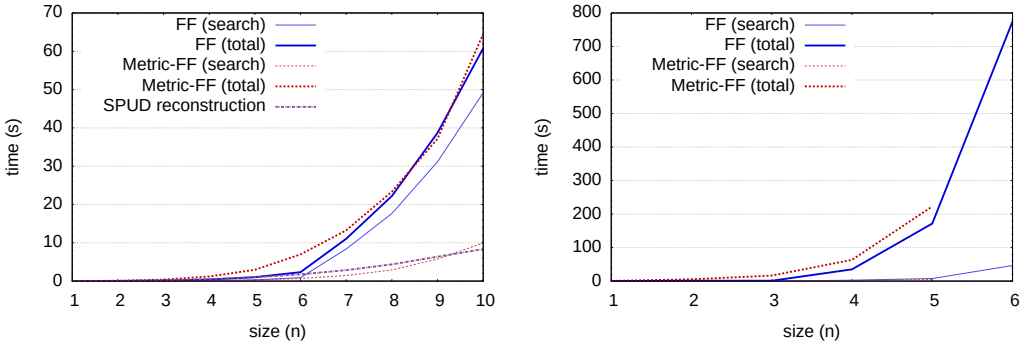


Figure 7: Results for the XTAG experiment, at $k = 1$ and $k = 2$.

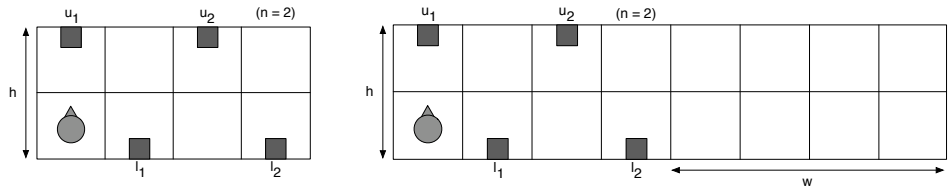
by one or two orders of magnitude. As a consequence, even our relatively naive Java implementation of GraphPlan outperforms them all in terms of total runtime, because it only computes instances by need. Although FF is consistently much faster as far as pure search time is concerned, our results indicate that FF's performance is much more sensitive to the domain size: if we fix $n = 1$, FF takes 27 milliseconds to compute a plan at $m = 1$, but 4.4 seconds to compute the same plan at $m = 10$. By comparison, our GraphPlan implementation takes 20 ms at $m = 1$ and still only requires 22 ms at $m = 10$.

4.2. Experiment 2: Sentence generation (XTAG)

The first experiment already gives us some initial insights into the appropriateness of planning for the sentence generation domain: On the examples we looked at, the search times were quite acceptable, but FF and SGPLAN spent a lot of time on the initial grounding step. However, one weakness of this experiment is that it uses a tiny grammar, consisting of just those 12 lexicon entries that are needed for the experiment. While the grounding problem can only get worse with larger grammars, the experiment by itself does not allow us to make clear statements about the search efficiency. To address this problem, we ran a second sentence generation experiment. This time, we used the XTAG Grammar (XTAG Research Group 2001), a large-scale TAG grammar of English. XTAG contains lexicon entries for about 17,000 uninflected words using about 1100 different elementary trees. Although XTAG does not contain semantic information, it is possible to automatically equip the lexicon entries with inferred semantic representations based on the words in the lexicalized elementary trees. The result is a highly ambiguous grammar: The most ambiguous word, "ask", is the anchor of 314 lexicon entries.

In our experiment, we were especially interested in two questions. First, how would the planners handle the search problem involved in generating with such a large and ambiguous grammar? Second, would it be harder to generate sentences containing verbs with multiple arguments, given that verbs with more arguments lead to actions with more parameters and therefore more instances? To answer these questions, we generated sentences of the form "S and S and ... and S", where each S was a sentence; we called the number of sentences in the conjunction n. Each S was a sentence of the form "the businessman sneezes", "the businessman admires the girl", or "the businessman gives the girl the book"—that is, they varied in the number k of syntactic arguments the verb expects (1 for the intransitive verb "sneeze", 2 for the transitive verb "admire", and 3 for the ditransitive verb "give"). This means that the output sentence for parameters n and k contained $n(2k + 2) - 1$ words. The instances were set up in such a way that the generation of referring expressions was trivial, so this experiment was purely a surface realization task. To achieve reasonable performance, we only generated planning operators for those elementary trees for which all predicate symbols in the semantic representation also appeared in the knowledge base.

Fig. 7 reports the runtimes we measured in this experiment for FF, Metric FF, and the SPUD



(a) Minimal GIVE world (b) GIVE world with extra grid cells

FIGURE 8: Experimental GIVE world configurations.

reimplementation. We do not report runtimes for SGPLAN, because we could not recompile SGPLAN as a 64-bit binary, and the 32-bit version ran out of memory very quickly. We also do not report runtimes for our Java implementation of GraphPlan, because it was unusably slow for serious problem instances: For $k = 1$ and $n = 3$, it already took over two minutes, and it exceeded its memory limit of 16 GB for $n > 3$. This may be a limitation of our naive implementation rather than the GraphPlan algorithm itself.

Nonetheless, there are a number of observations we can make in this experiment. First of all, the experiment confirms that FF's Enforced Hill-Climbing search strategy works very well for the sentence generation task: Although we are now generating with a large grammar, FF generates a 39-word sentence ($k = 1, n = 6$) in under a second of search time. This level of efficiency is a direct result of using this particular search strategy: For $k = 1$ and $n > 6$, FF 2.3 (but not Metric-FF) fell back to the best-first search strategy, which causes a dramatic loss of search efficiency. It is also encouraging that Metric-FF still performs comparably to SPUD in terms of pure search time. We believe that by FF's technique of evaluating actions by estimating the distance to a goal state for the relaxed problem essentially picks out the same evaluation function as SPUD's domain-specific heuristic, and the enforced hill-climbing strategy needs to backtrack very little in this domain and thus performs similarly to SPUD's greedy search. However, SPUD's incompleteness manifests itself in this experiment by its inability to find any plan for $k > 1$ and $n > 1$, whereas FF and its variants still (correctly) find these plans.

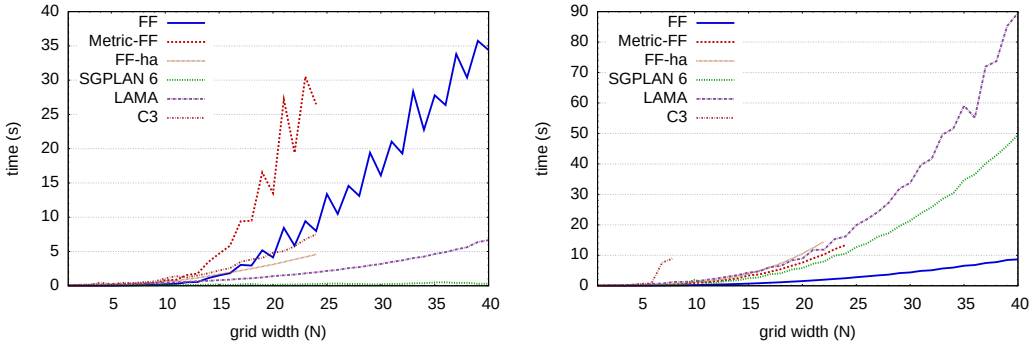
Second, FF's runtime is still dominated by the preprocessing stage. For instance, Metric-FF spends about 10 seconds on search for $k = 1, n = 10$, compared to its total runtime of about 65 seconds. This effect becomes more pronounced as we increase k: For $k = 2$, we reach 65 seconds of total runtime at $n = 4$, but here Metric-FF only spends about a second on search. For $k = 3$, neither FF nor Metric-FF were able to solve any of the input instances within their memory limit. This is consistent with the observation that the planning operators for the verbs have $k + 2$ parameters (see Fig. 3), and thus the number of action instances grows by a factor of the universe size every time we increase k by one. A planner which computes all ground instances of the operators thus takes exponential time in k for preprocessing.

4.3. Experiment 3: Minimal GIVE worlds

We now turn our attention to a set of experiments arising from the GIVE domain. Besides using many of the planners from the previous set of experiments (FF, Metric-FF, and SGPLAN), we also expand our testing to include the FF(h_a) (Keyder and Geffner 2008), LAMA (Richter and Westphal 2008), and C^3 (Lipovetzky et al. 2008) planners. Each of these additional planners competed in the deterministic "sequential, satisficing" track of the 2008 International Planning Competition; all planners performed well on the competition domains, with LAMA the overall winner of the track.[7]

In the first GIVE experiment, we construct a series of grid worlds, similar to the one illustrated in Figure 8(a). These worlds consist of a $N = 2n$ by h grid of positions, such that there are buttons at positions $(2i - 1, 1)$ and $(2i, h)$ for $1 \leqslant i \leqslant n$. The player starts in position $(1, 1)$ and must press all the buttons to successfully complete the game. (The actions in this domain are similar to the PDDL actions in Figure 5.) We consider two variants of this problem in our tests. In the *unordered*

[7]See `http://ipc.informatik.uni-freiburg.de/` for details of the 2008 IPC.



(a) Unordered (b) Ordered

Figure 9: Results for the unordered and ordered minimal GIVE domains with grid height $h = 20$. The horizontal axis is the grid width, N. The vertical axis is the total runtime in seconds.

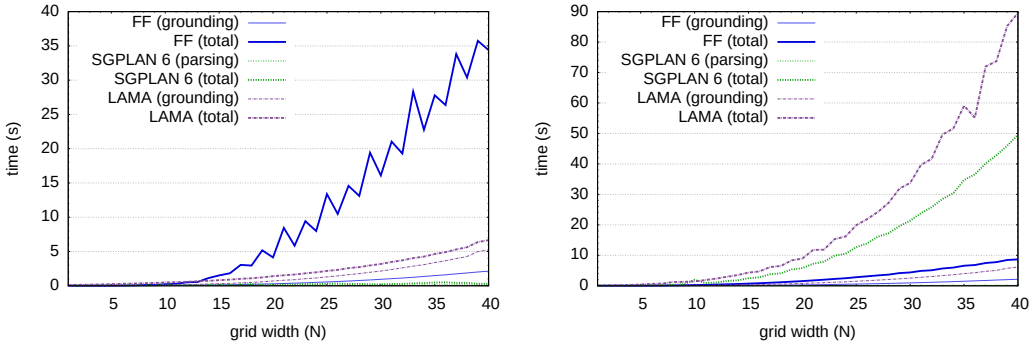
problem, the player is permitted to press the buttons in any order to successfully achieve the goal. In the *ordered* version of the problem, the player is unable to initially move to any grid cell containing a button, except for the cell containing the first button, u_1. Pressing u_1 releases the position of the next button, l_1, allowing the player to move into this cell. Similarly, pressing button l_1 frees button u_2, and so on. The end result is a set of constraints that forces the buttons to be pressed in a particular order to achieve the goal. As a concrete example, the following is a minimal plan (in either variant of the problem) for the case of a 2 by 2 grid with 2 buttons (i.e., $n = 1$, $h = 2$):

(1) move(pos_1_1, pos_1_2, north),
(2) manipulate-button-off-on(u1, pos_1_2),
(3) turn-right(north, east),
(4) move(pos_1_2, pos_2_2, east),
(5) turn-right(east, south),
(6) move(pos_2_2, pos_2_1, south),
(7) manipulate-button-off-on(l1, pos_2_1).

Results for the $h = 20$ case, with the grid width N ranging from 1 to 40, are shown in Figure 9. In the unordered case (Figure 9(a)), the most obvious result is that some of the planners tested—Metric-FF, FF(h_a), and C^3—are unable to solve any problems beyond $N = 24$ on our experimentation machine within the memory limit of 2 GB.[8] While FF, LAMA, and SGPLAN are able to solve all problem instances up to $N = 40$, the total running time varies greatly between these planners. For instance, FF takes almost 35 seconds to solve the $N = 40$ problem, while LAMA takes around 6.5 seconds. SGPLAN shows impressive performance on $N = 40$, generating a 240 step plan in well under a second. In the ordered case (Figure 9(b)), we again have the situation where Metric-FF, FF(h_a), and C^3 are unable to solve all problem instances. Furthermore, both SGPLAN and LAMA, which performed well on the unordered problem, now perform much worse than FF: FF takes 39 seconds for the $N = 40$ case, while SGPLAN takes 50 seconds and LAMA takes 90 seconds. In real NLG systems, where response time is essential, running times over a few seconds are unacceptable.

Preprocessing time (parsing, grounding, etc.) generally plays less of a role in GIVE, compared with the sentence generation domain; however, its effects still contribute significantly to the overall running time of a number of planners. Figure 10 shows the grounding time for FF, LAMA, and SGPLAN on the minimal GIVE problems, compared with the total running time. In the unordered variant of the minimal GIVE domain (Figure 10(a)), the grounding time in LAMA and SGPLAN accounts for a significant fraction of the total runtime: SGPLAN spends around 40% of its total

[8] All runtimes in Sections 4.3 and 4.4 were measured on a single core of an Intel Xeon CPU running at 3GHz, under Linux. All runtimes are averaged over three runs of the planners. Only 32-bit versions of the planners were used for testing in each case.



(a) Unordered (b) Ordered

FIGURE 10: Comparison of the total running time and grounding time for selected planners in the $h = 20$ minimal GIVE domain. The horizontal axis is the grid width, N. The vertical axis is the total runtime in seconds.

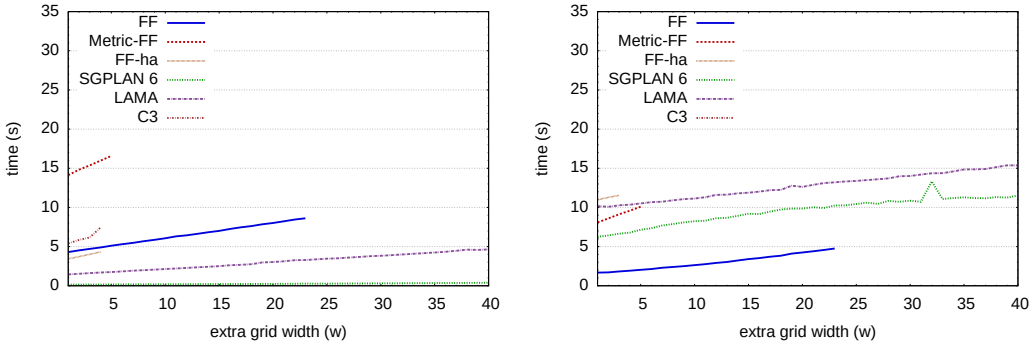
runtime on preprocessing; for LAMA, this number rises to at least 80% for our test problems. For FF, the preprocessing time is much less important than the search time, especially for large problem instances. In the ordered case (Figure 10(b)), the actual time spent on preprocessing is essentially unchanged from the unordered case, and search time dominates the total runtime for all three planners. Overall, however, FF is now much better at controlling the search, compared with the other planners and its performance on the unordered variant of the problem.

4.4. Experiment 4: GIVE worlds with extra grid cells

In our last set of experiments, we vary the structure of the GIVE world in order to judge the effect that universe size has on the resulting planning problem. Starting with the GIVE world described in Experiment 3, we extend the world map by adding another w by h empty cell positions to the right of the minimal world, as shown in Figure 8(b). These new positions are not actually required in any plan, but extend the size of the state space and approximate the situation in the actual GIVE domain where most grid positions are never used. We leave the initial state and goal untouched and, again, consider both unordered and ordered variants of the problem.

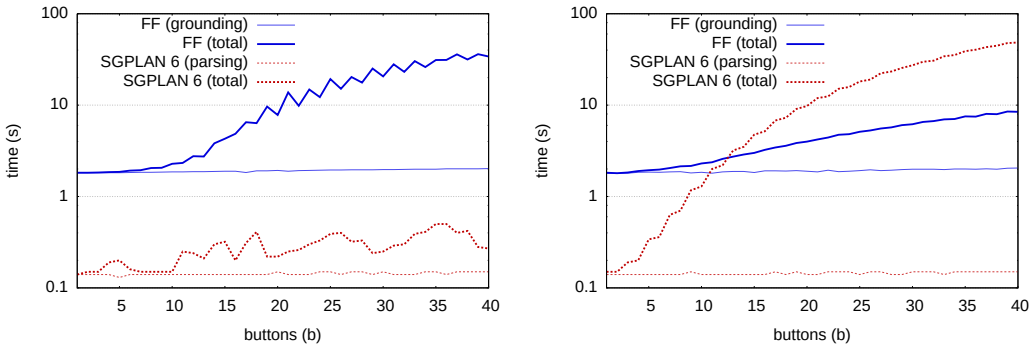
Results for the $h = 20$, $n = 10$ case with w ranging from 1 to 40 are shown in Figure 11. As in Experiment 3, a number of planners again fail to solve all the problems: Metric-FF, FF(h_a), and C^3 solve only a few instances, while FF only scales to $w = 23$. In the unordered version of the domain, SGPLAN easily solves inputs beyond $w = 40$ in well less than a second. LAMA is also reasonably successful on these problems; however, its runtimes grow more quickly than SGPLAN, with LAMA taking almost 5 seconds to solve the $w = 40$ problem instance. In the ordered case, we again see behaviour similar to that of Experiment 3: for the problem instances FF is able to solve, it performs significantly better than LAMA and SGPLAN. (SGPLAN's long term runtime appears to be growing at a slower rate than FF's, and so even if FF could be scaled to larger problem instances, it seems possible that SGPLAN might overtake FF as the better performer.) However, the overall planning times for most of these instances are concerning since times over a couple seconds will negatively affect the response time of an NLG system, which must react in real time to user actions.

Finally, we also performed a set of experiments designed to investigate the tradeoff between grounding time and search time on certain grid configurations. For these experiments, we initially fixed the size of the grid and then varied the number of buttons b in the world, thereby creating a series of "snapshots" of particular extra-cell GIVE domains. Figure 12 shows the results of these experiments for the FF and SGPLAN planners, for a fixed size grid of height 20 and width 40, and the number of buttons b ranging from 1 to 40. In each case, the amount of time a planner spends on grounding is relatively unchanged as we vary the number of buttons in a grid, while the search time continues to rise (sometimes quite dramatically), as b increases (we saw a similar effect for other grid



(a) Unordered (b) Ordered

FIGURE 11: Results for the unordered and ordered GIVE domains with $h = 20$ and $n = 10$. The horizontal axis is the extra grid width w. The vertical axis is the total runtime in seconds.



(a) Unordered (b) Ordered

FIGURE 12: Results for the GIVE domains with a fixed grid size of height 20 and width 40. The horizontal axis is the number of buttons b. The vertical axis is the runtime in seconds (log scale).

configurations we tried). This observation has important consequences for the design of our GRID worlds: changing the underlying domain structure, even minimally, may result in significant—and often unexpected—performance differences for the planners that must operate in these domains.

5. DISCUSSION

We can draw both positive and negative conclusions from our experiments about the state of planning for modern NLG applications. On the one hand, we found that modern planners are very good at dealing with the *search* problems that arise in the NLG-based planning problems we investigated. In the sentence generation domain, FF's Enforced Hill-Climbing strategy finds plans corresponding to 25-word sentences in about a second. It is hard to compare this number to a baseline because there are no shared benchmark problems, but FF's search performance is similar to that of a greedy, incomplete special-purpose algorithm, and competitive to other sentence generators as well. Thus research on search strategies for planning has paid off; in particular, the Enforced Hill-Climbing heuristic outperforms the best-first strategy to which FF 2.3 switches for some problem instances. Similarly, SGPLAN's performance on the GIVE domain is very convincing and fast enough for many instances of this application.

On the other hand, each of the off-the-shelf planners we tested spent substantial amounts of time

on preprocessing. This is most apparent in the sentence generation domain, where the planners spent almost their entire runtime on grounding the predicates and operators for some problem instances. This effect is much weaker in the GIVE domain, which has a much smaller number of operators and less interactions between the predicates in the domain. However, our GIVE experiments also illustrate that altering the structure of a domain, even minimally, can significantly change a planner's performance on a problem. For instance, in some of our GIVE experiments with extra grid positions, increasing the number of buttons in the world, while keeping the dimensions of the grid fixed, resulted in a significantly larger search time while the preprocessing time remained essentially unchanged.

While the GIVE domain can be defined in such a way that the number of operators is minimized, this is not possible for an encoding of a domain in which the operators model the different communicative actions that the NLG system can use. For instance, in the sentence generation domain, the XTAG planning problem for $k = 2$ and $n = 5$ consists of about 1000 operators for the different lexicon entries for all the words in the sentence, some of which take four parameters. It is not unrealistic to assume a knowledge base with a few hundred individuals. All this adds up to trillions of ground instances: a set which is completely infeasible to compute naively.

Of course, it would be premature to judge the usefulness of current planners as a whole, based on just two NLG domains. Nevertheless, we believe that the structure of our planning problems, which are dominated by large numbers of operators and individuals, is typical of NLG-related planning problems as a whole. This strongly suggests that while current planners are able to manage many of the search problems in the domains we looked at, they are still unusable for practical NLG applications because of the time they spend on preprocessing. In other words, the state of generation-as-planning research is still not in a much better position than it was in the 1980s.

We are also aware that the time a planner invests in preprocessing can pay off during search, and that such techniques have been invaluable in improving the overall running time of modern planners. However, we still suggest that the inability of current planners to scale to larger domains limits their usefulness for applications beyond NLG as well. Furthermore, we feel that the problem of preprocessing receives less research attention than it deserves: if the problem is scientifically trivial then we challenge the planning community to develop more efficient implementations that only ground operators by need; otherwise, we look forward to future publications on this topic. To support this effort, we offer our planning domains as benchmarks for future research and competitions.[9]

Finally, we found it very convenient that the recent International Planning Competitions provide a useful entry point for selecting and obtaining current planners. Nevertheless, our experiments exposed several bugs in the planners we tested, which required us to change their source code to make them scale to our inputs. We also found that different planners that solve the same class of planning problems (e.g., STRIPS, ADL, etc.) sometimes differ in the variants of PDDL that they support. These differences range from fragments of ADL that can be parsed, to sensitivity to the order of declarations and the use of "objects" rather than "individuals" as the keyword for declaring the universe. We propose that the case for planning as a mature technology with professional-quality implementations could be made more strongly if such discrepancies were harmonized.

6. CONCLUSION

In this paper, we investigated the usefulness of current planning technology to natural language generation, an application area with a long tradition of using automated planning that has recently experienced renewed interest from NLG researchers. In particular, we evaluated the performance of several off-the-shelf planners on a series of planning domains that arose in the context of sentence generation and situated instruction generation.

Our results were mixed. While some of the planners we tested—in particular, FF and SGPLAN—did an impressive job of controlling the complexity of the search, we also found that all the planners we tested spent too much time on preprocessing to be useful. For instance, in the sentence generation domain, FF spent 90% of its runtime on computing the ground instances of the planning operators;

[9]The PDDL problem generators for our NLG domains are available at `http://www.coli.uni-saarland.de/~koller/projects/crisp`.

in the instruction-giving domain, which is very similar to Gridworld, a similar effect happened for certain combinations of grid sizes and buttons. As things stand, we found that this overly long preprocessing time makes current planners an inappropriate choice for NLG applications, in any but the smallest problem instances. Users who come to planning from outside the field, such as NLG researchers, treat planners as black boxes. This means that search efficiency alone is not helpful when other modules of the planner are slow. From this perspective, we propose that the planning community should spend some attention on optimising the preprocessing component of the problem with similar vigour as the search itself. In particular, we propose that one line of research might be to investigate planning algorithms that do not rely on grounding out all operators prior to the search, but instead selectively perform this operation when needed.

NLG and planning have a long history in common. The recent surge in NLG-as-planning research presents valuable opportunities for both disciplines. Clearly, NLG researchers who apply planning technology will benefit directly from any improvements in planner efficiency. Conversely, NLG may also be a worthwhile application area for planning researchers to keep in mind. Domains like GIVE highlight certain challenges, such as plan execution monitoring and plan presentation (i.e., summarisation and elaboration), but also offer a platform on which such technologies can be evaluated in experiments with human users. Furthermore, although we have focused on classical planning problems in this work, research related to reasoning under uncertainty, resource management, and planning with knowledge and sensing, can also be investigated in these settings. As such, we believe our domains would provide interesting challenges for planners entered in future editions of the IPC.

Acknowledgements

This work arose in the context of the Planning and Language Interest Group at the University of Edinburgh. The authors would like to thank all members of this group, especially Héctor Geffner and Mark Steedman, for interesting discussions. We also thank our reviewers for their insightful and challenging comments. This work was supported by the DFG Research Fellowship "CRISP: Efficient integrated realization and microplanning", the DFG Cluster of Excellence "Multimodal Computing and Interaction", and by the European Commission through the PACO-PLUS project (FP6-2004-IST-4-27657).

REFERENCES

Appelt, D., 1985. Planning English Sentences. Cambridge University Press, Cambridge, England, 171 pp.

Benotti, L., 2008. Accommodation through tacit sensing. In Proceedings of the 12th Workshop on the Semantics and Pragmatics of Dialogue. London, United Kingdom, pp. 75–82.

Blum, A. and M. Furst, 1997. Fast planning through graph analysis. Artificial Intelligence, **90**:281–300.

Brenner, M. and I. Kruijff-Korbayová, 2008. A continual multiagent planning approach to situated dialogue. In Proceedings of LonDial.

Byron, D., A. Koller, K. Striegnitz, J. Cassell, R. Dale, J. Moore, and J. Oberlander, 2009. Report on the first nlg challenge on generating instructions in virtual environments (give). In Proceedings of the 12th European Workshop on Natural Language Generation. Athens.

Hoffmann, J., 2002. Extending FF to numerical state variables. In Proceedings of the 15th European Conference on Artificial Intelligence (ECAI-02). pp. 571–575.

Hoffmann, J. and S. Edelkamp, 2005. The deterministic part of IPC-4: An overview. Journal of Artificial Intelligence Research, **24**:519–579.

Hoffmann, J. and B. Nebel, 2001. The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research, **14**:253–302.

Hovy, E., 1988. Generating natural language under pragmatic constraints. Lawrence Erlbaum Associates, Hillsdale, NJ, USA, 224 pp.

Hsu, C. W., B. W. Wah, R. Huang, and Y. X. Chen, 2006. New features in SGPlan for handling soft constraints and goal preferences in PDDL 3.0. In Proceedings of the Fifth International

Planning Competition, 16th International Conference on Automated Planning and Scheduling. The English Lake District, Cumbria, United Kingdom, pp. 39–41.

JOSHI, A. and Y. SCHABES, 1997. Tree-Adjoining Grammars. In Handbook of Formal Languages, edited by G. Rozenberg and A. Salomaa, Springer-Verlag, Berlin, Germany, volume 3. pp. 69–123.

KEYDER, E. and H. GEFFNER, 2008. The ff(h_a) planner for planning with action costs. In Proceedings of the Sixth International Planning Competition.

KOLLER, A. and M. STONE, 2007. Sentence generation as planning. In Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics. Prague, Czech Republic, pp. 336–343.

KOLLER, A. and K. STRIEGNITZ, 2002. Generation as dependency parsing. In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics. Philadelphia, PA, USA, pp. 17–24.

LIPOVETZKY, N., M. RAMIREZ, and H. GEFFNER, 2008. C3: Planning with consistent causal chains. In Proceedings of the Sixth International Planning Competition.

MCDERMOTT, D. and THE AIPS-98 PLANNING COMPETITION COMMITTEE, 1998. PDDL – The Planning Domain Definition Language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 27 pp.

PERRAULT, C. R. and J. F. ALLEN, 1980. A plan-based analysis of indirect speech acts. American Journal of Computational Linguistics, **6**(3–4):167–182.

REITER, E. and R. DALE, 2000. Building Natural Language Generation Systems. Cambridge University Press, Cambridge, England, 248 pp.

RICHTER, S. and M. WESTPHAL, 2008. The LAMA planner: Using landmark counting in heuristic search. In Proceedings of the Sixth International Planning Competition.

STEEDMAN, M. and R. P. A. PETRICK, 2007. Planning dialog actions. In Proceedings of the Eighth SIGdial Workshop on Discourse and Dialogue. Antwerp, Belgium, pp. 265–272.

STOIA, L., D. M. SHOCKLEY, D. K. BYRON, and E. FOSLER-LUSSIER, 2008. SCARE: A situated corpus with annotated referring expressions. In Proceedings of the 6th International Conference on Language Resources and Evaluation (LREC 2008).

STONE, M., C. DORAN, B. WEBBER, T. BLEAM, and M. PALMER, 2003. Microplanning with communicative intentions: The SPUD system. Computational Intelligence, **19**(4):311–381.

STONE, M. and B. WEBBER, 1998. Textual economy through close coupling of syntax and semantics. In Proceedings of the Ninth International Workshop on Natural Language Generation. pp. 178–187.

TOVEY, C. and S. KOENIG, 2000. Gridworlds as testbeds for planning with incomplete information. In Proceedings of the 17th National Conference on Artificial Intelligence. Austin, TX, USA, pp. 819–824.

XTAG RESEARCH GROUP, 2001. A lexicalized tree adjoining grammar for english. Technical Report IRCS-01-03, IRCS, University of Pennsylvania. `ftp://ftp.cis.upenn.edu/pub/xtag/release-2.24.2001/tech-report.pdf`.

YOUNG, R. M. and J. D. MOORE, 1994. DPOCL: a principled approach to discourse planning. In Proceedings of the Seventh International Workshop on Natural Language Generation. Kennebunkport, Maine, USA, pp. 13–20.