Waking Up a Sleeping Rabbit: On Natural-Language Sentence Generation with FF

Alexander Koller
Saarland University, Germany
koller@mmci.uni-saarland.de

Jörg Hoffmann
INRIA, France
joerg.hoffmann@inria.fr

Abstract

We present a planning domain that encodes the problem of generating natural language sentences. This domain has a number of features that provoke fairly unusual behavior in planners. In particular, hitherto no existing automated planner was sufficiently effective to be of practical value in this application. We analyze in detail the reasons for ineffectiveness in FF, resulting in a few minor implementation fixes in FF's preprocessor, and in a basic reconfiguration of its search options. The performance of the modified FF is up to several orders of magnitude better than that of the original FF, and for the first time makes automated planners a practical possibility for this application. Beside thus highlighting the importance of preprocessing and automated configuration techniques, we show that the domain still poses several interesting challenges to the development of search heuristics.

Introduction

Natural language generation (NLG) – the problem of computing a sentence or text that communicates a given piece of information – has a long-standing connection to automated planning (Perrault and Allen 1980). In the past few years, the idea has been picked up again in the NLG community, both to use planning as an approach for modeling language problems (e.g., (Steedman and Petrick 2007)) and to tackle the search problems in NLG by using techniques from planning (Koller and Stone 2007).

Existing planners are relatively good at solving modest instances of the planning problems that arise in this context. However, Koller and Petrick (2010) recently showed that as the planning problem instances from Koller and Stone (2007) scale up, off-the-shelf planners become slow – indeed, too slow to be practically useful in NLG applications. They showed this for SGPlan and several variants of FF (Hoffmann and Nebel 2001); we show it herein also for LAMA (Richter, Helmert, and Westphal 2008). Koller and Petrick noted that there are major sources of inefficiency both in the search itself and in the commonly used preprocesses; they identified several features of the domain (e.g. comparatively large predicate and operator arities) that diverge from most IPC benchmarks.

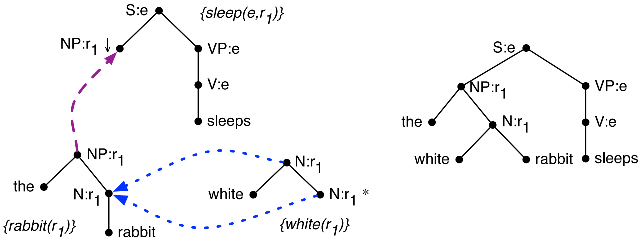

Figure 1: Derivation of "The white rabbit sleeps."

In this paper, we revisit this situation, and analyze the sources of inefficiency in detail for FF. Based on the analysis, we modify FF and obtain dramatically better performance that, for the first time, makes automated planners a realistic possibility for NLG applications. To achieve this quantum leap, all that is needed are a few minor implementation fixes in FF's preprocessor, and basic reconfiguration of its search options. While these changes certainly are no contribution to planning in their own right, we consider it a valuable – and slightly unsettling – lesson learned that such banalities can make the difference between success and failure. They can hold up progress in planning applications for years, unless users of a planner have direct access to its developer. We interpret this as a strong call for more research on automated configuration techniques, e.g. along the lines of Hutter et al. (2007). The detrimental effect of preprocessing details also sheds a light on the importance of this often neglected part of the planning process.

Apart from these insights, our domain remains an interesting challenge for the development of better heuristics. There still are realistic instances of the NLG planning domain on which our modified FF is not practical. Overcoming this weakness would entail overcoming fundamental weaknesses of FF's goal ordering techniques and the relaxed plan heuristic – variants of which are used in many planners today.

Sentence generation as planning

We consider a domain pertaining to the generation of natural language sentences based on tree-adjoining grammar (TAG; (Joshi and Schabes 1997)). This is a standard problem in computational linguistics, which we sketch by example; see (Stone et al. 2003; Koller and Stone 2007) for details.

A TAG grammar consists of a finite set of *elementary trees*, each of which contains one or more words; the left

sleeps(u, u_1, u_n, x_0, x_1)**:**
Precond: $\mathsf{subst}(\mathsf{S}, u) \wedge \mathsf{ref}(u, x_0) \wedge \mathsf{sleep}(x_0, x_1)$
$\wedge\, \mathsf{current}(u_1) \wedge \mathsf{next}(u_1, u_n)$
Effect: $\neg\mathsf{subst}(\mathsf{S}, u) \wedge \mathsf{expressed}(\mathsf{sleep}, x_0, x_1)$
$\wedge\, \mathsf{subst}(\mathsf{NP}, u) \wedge \mathsf{ref}(u_1, x_1)$
$\wedge\, \neg\mathsf{current}(u_1) \wedge \mathsf{current}(u_n)$
$\wedge\, \forall y. y \neq x_1 \rightarrow \mathsf{distractor}(u_1, y)$

rabbit(u, x_0)**:**
Precond: $\mathsf{subst}(\mathsf{NP}, u) \wedge \mathsf{ref}(u, x_0) \wedge \mathsf{rabbit}(x_0)$
Effect: $\neg\mathsf{subst}(\mathsf{NP}, u) \wedge \mathsf{canadjoin}(\mathsf{N}, u)$
$\wedge\, \forall y. \neg\mathsf{rabbit}(y) \rightarrow \neg\mathsf{distractor}(u, y)$

white(u, x_0)**:**
Precond: $\mathsf{canadjoin}(\mathsf{N}, u) \wedge \mathsf{ref}(u, x_0) \wedge \mathsf{rabbit}(x_0)$
Effect: $\forall y. \neg\mathsf{white}(y) \rightarrow \neg\mathsf{distractor}(u, y)$

Figure 2: Actions for generating “The white rabbit sleeps.”

of Fig. 1 shows three trees for “the rabbit”, “sleeps”, and “white”. Trees can be combined using the operations of *substitution* (purple dashed arrow in the figure) and *adjunction* (blue dotted arrows) to form larger trees. The end result of a grammatically correct derivation is a tree all of whose leaves are labeled with words, as on the right of Fig. 1. We can read off a sentence from such a tree from left to right.

To use TAG grammars for generation (Stone et al. 2003), we assume a set of ground atoms expressing the information we want the sentence to convey, such as $\{\mathsf{sleep}(e, r_1)\}$, and a knowledge base that contains all ground atoms we know to be true; say, $\{\mathsf{sleep}(e, r_1), \mathsf{rabbit}(r_1), \mathsf{rabbit}(r_2), \mathsf{white}(r_1), \mathsf{brown}(r_2)\}$. We then add variables ranging over constants from the knowledge base to the nodes of each elementary tree, and equip every elementary tree with a set of atoms over these variables to encode the meaning this tree expresses (printed in italics in the figure). Fig. 1 already shows specific instances of the elementary trees from the grammar, in which constants have been substituted for these variables. The derivation in the figure conveys the intended meaning – in particular, that r_1 sleeps. Crucially, it also describes uniquely who does the sleeping: The sentence “the rabbit sleeps” would not have done this, as “the rabbit” could be understood either as r_1 or r_2. We say that r_2 is a *distractor* for the subject of the sentence, which is removed by adjoining the tree for “white”. In short, the sentence generation problem consists in computing a grammatically correct TAG derivation from instances of the elementary trees in the grammar that conveys the intended meaning and describes all referents uniquely.

Sentence generation as a planning problem

The TAG sentence generation problem can be encoded as a planning problem (Koller and Stone 2007). The key idea is that each operator encodes the addition of an elementary tree to the derivation; the syntactic and semantic requirements and effects of doing this are captured in the operator's precondition and effect.

More precisely, each operator has a parameter u representing the syntax node into which the elementary tree is substituted or adjoined, and a parameter u_i for each substitution node. There are also parameters $x_0, \dots, x_k$ for the variables in the semantic representation of the elementary tree; x_0 is the variable that occurs at the root of the tree. The planning state encodes a list of possible node names: $\mathsf{current}(u)$ expresses that u is the next node name that should be picked when a new substitution node is created, and $\mathsf{next}(u, v)$ says that the next node after u is v. These atoms are used to select names for the substitution nodes that are introduced by adding an elementary tree; the parameter u_n represents the node that will be current after the addition.

The atom $\mathsf{subst}(A, u)$ expresses that we need to still substitute something into the substitution node u with label A; the initial state contains an atom $\mathsf{subst}(\mathsf{S}, \mathsf{root})$ where S stands for “sentence” and root is an arbitrary name for the root of the derivation. The mapping from nodes to semantic constants is maintained using the ref atoms; the initial state for generating a sentence about the event e contains an atom $\mathsf{ref}(\mathsf{root}, e)$. Finally, we keep track of the uniqueness of referring expressions in the $\mathsf{distractor}$ atoms: $\mathsf{distractor}(u, a)$ expresses that the expression at syntax node u could still be misunderstood as a.

The example generation problem above translates into a planning problem whose domain is shown in Fig. 2. The initial state of the problem encodes the generation problem's knowledge base; it contains atoms $\mathsf{rabbit}(r_1)$, $\mathsf{sleep}(e, r_1)$, etc. The goal requires syntactic completeness as $\forall u \forall A \neg\mathsf{subst}(A, u)$ and unique reference as $\forall u \forall x \neg\mathsf{distractor}(u, x)$; it also specifies the semantic representation we want to express in an atom $\mathsf{expressed}(\mathsf{sleep}, e, r_1)$.

A minimal plan for the example problem is $\mathsf{sleeps}(\mathsf{root}, n_1, n_2, e, r_1)$; $\mathsf{rabbit}(n_1, r_1)$; $\mathsf{white}(n_1, r_1)$. This plan can be automatically decoded into the derivation shown in Fig. 1. The first two steps of the plan alone would not be a correct plan because they leave an atom $\mathsf{distractor}(n_1, r_2)$ in the state, which contradicts the goal that all distractors have been eliminated.

FF Pitfalls and Fixes

The basic observation of FF's inefficiency in the NLG domain was made before (Koller and Petrick 2010); here we examine its causes in detail. For some of the observed pitfalls, we are able to provide simple fixes. Others pose serious challenges to planning research.

Preprocessing

While preprocessing the ADL formulas, FF has a subroutine that checks the grounded formulas for tautologies. This involves a loop checking all pairs of sub-formulas within every conjunction and disjunction, testing whether they are identical/identical modulo their sign. For example, if we find ϕ and $\neg\phi$ within a disjunction, then the disjunction is replaced by TRUE. We haven't tested whether this subroutine has a notable effect in other benchmarks. In our domain, the only effect it has is to take a huge amount of runtime on the long goal conjunctions stating that we do not want any open substitution nodes or distractors. The fix is to switch this subroutine off. This trivial operation alone can render previously infeasible instances (preprocessing time $>$ 15 minutes) harmless (instance solved in 6 seconds *total* time).

FF's preprocessor contains a very small imprudence that apparently didn't surface in other domains, but that does in ours. FF distinguishes operators into "easy" (DNF precondition) and "hard" (non-DNF precondition), which go through different instantiation procedures. The procedure for hard operators uses a data structure encompassing an integer for every possible instantiation of all predicates. Since this does not take into account reachability, it is prohibitively huge in our domain. Now, we actually have no hard operators. But FF builds the data structure anyway, which exhausts memory and causes a segmentation fault on serious instances.

Search Techniques

FF's preprocessing pitfalls are certainly baffling, but easily fixed. We now turn our attention to some of FF's search techniques, variations of which are used in many planners, and which are very much not easily fixed. Our solution for the moment are simple configuration changes.

First, we consider Koehler and Hoffmann's (2000) "reasonable goal orderings", variants and extensions of which are in prominent use today, e.g. in LAMA (Richter, Helmert, and Westphal 2008). FF as used in the 2000 competition approximates reasonable orders as a preprocess and then partitions the goal set into a *goal agenda*, a sequence of goal subsets. Each subset is posed to the planner in isolation. The final plan is the concatenation of the plans for all entries.

The goal agenda is completely off-target in our domain. The most striking point concerns the interaction between the goal to create a sentence ($\neg\mathsf{subst}(S, \mathsf{root})$ in the rabbit example), and the goal to express a certain meaning ($\mathsf{expressed}(\mathsf{sleep}, e, r_1)$). The goal agenda orders the former before the latter because it detects that, once a meaning is expressed, the sentence creation has started and it is not possible to start anew anymore. Unless the inverse also holds,[1] $\mathsf{subst}(S, \mathsf{root})$ is alone in the first goal agenda entry. Consequently, the planner decides to generate *any* sentence, with some arbitrary meaning. If it happens to be the wrong sentence, the planner is stuck in a dead-end. It is not clear to us, at this point, how goal ordering techniques could be modified so that they return more sensible results.

In our improved FF, the goal agenda is simply switched off. Having thus disposed of goal orderings, let us turn our attention to the relaxed plan heuristic. This meets its Waterloo in the unusual life-cycle of facts encoding substitution nodes and distractors. These facts are required to be false in the goal, and most of them are initially false. However, any valid plan must make many of them true at some intermediate time point (e.g. distractors arise depending on the decisions taken by the planner). Hence, from the perspective of the relaxed plan heuristic, from one state to the next whole classes of goals suddenly appear "out of the blue".

Consider again the rabbit example. In the initial state, the relaxed plan has length 1 because we only need to achieve $\neg\mathsf{subst}(S, \mathsf{root})$ and $\mathsf{expressed}(\mathsf{sleep}, e, r_1)$, which is done by the single action "sleeps". However, that action has the effect $\forall y. y \neq x_1 \rightarrow \mathsf{distractor}(u_1, y)$, which introduces new distractors. Thus the relaxed plan for the second state contains *two* actions. In general (if there are n rabbits, say), the relaxed plan may become arbitrarily long. For FF's particular search algorithm, enforced hill-climbing (EHC), this means that *FF turns into a blind breadth-first search*: EHC starts at h value 1 and thereafter needs to solve the whole problem in order to find a (goal) state with lower h value.

More generally, from this example we immediately see that, in the terms of Hoffmann (2005), the exit distance from local minima under h^+ is unbounded. Also, we have unrecognized dead-ends, i.e., dead-end states that have a relaxed plan. An example is the initial state of a task that it unsolvable because not all distractors can be removed; its h value will still be 1. So the domain is in the most difficult class of Hoffmann's (2005) planning domain taxonomy. While it shares this property with several other challenging domains, turning EHC into a breadth-first search is unheard of.

Can we design heuristics that are better at dealing with the life-cycle of substitution nodes and distractors? It is easy to see that the problem of not noticing them in the initial state is present for most known heuristics (Helmert 2004; Richter, Helmert, and Westphal 2008; Karpas and Domshlak 2009; Helmert and Domshlak 2009; Cai, Hoffman, and Helmert 2009). The two notable exceptions are the h^m family, and abstraction heuristics. We ran a few tests with a variant of the example problem involving 10 rabbits. The Graphplan estimate for the initial state is 4, i.e., h^2 is better than h^+ here (although still far from the real goal distance of 14). With merge&shrink (Helmert, Haslum, and Hoffmann 2007), we obtained the estimate 11 for the initial state, which is much better. Still the search space with that latter heuristic remained fairly large (5200 nodes vs. 13800 nodes for blind search), and it is not clear how well the technique will scale to larger instances and instances with a more complex grammar. Certainly, interesting challenges remain.

The only FF technique that appears to be clearly useful here is its action pruning technique. While the relaxed plan is often much too short, it appears to always contain at least one of the actions that actually are useful. Given the above observations, we modified FF to directly run best-first search with helpful actions (as opposed to original FF, which switches to best-first with*out* helpful actions if EHC fails).

Experiments

To evaluate the effect of our changes on the efficiency of the planner on practical inputs, we ran a series of experiments using the XTAG Grammar (XTAG Research Group 2001), a large-scale tree-adjoining grammar of English, from which we selected all lexicon entries for seven words we used in the experiments. We set up inputs to generate sentences of the form "S_1 and S_2 and … and S_n", i.e. a conjunction of n sentences. Each sentence S_i is of the form "X admires Y". X and Y are unique referring expressions, such as "the rich businessman". For each experiment, we have a parameter d that specifies how many adjectives are needed to distinguish the referent uniquely – in particular, $d = 0$ means we can say "the businessman", and $d = 2$ means we must say "the rich sad businessman".

[1] The inverse holds if, in the given grammar, the only way to communicate a meaning is in the main clause – i.e., there are no subordinate clauses like "the sleeping rabbit".

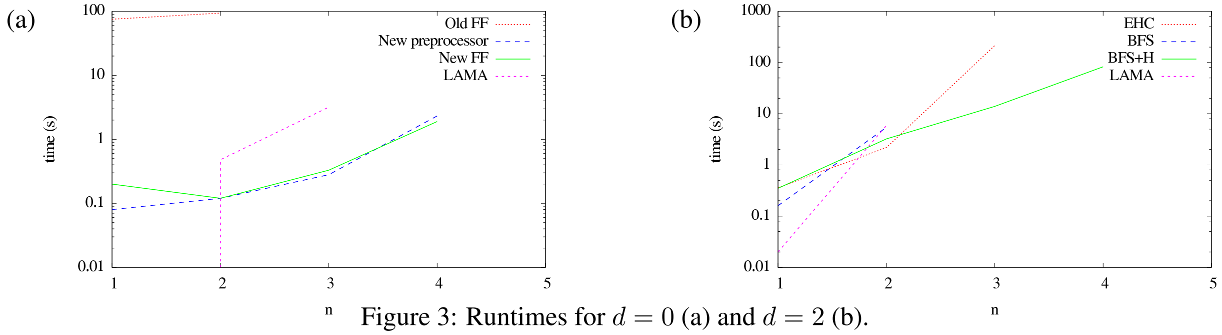

Figure 3: Runtimes for $d = 0$ (a) and $d = 2$ (b).

Consider first the results for $d = 0$, shown in Fig. 3 (a); the lines end where the planner exceeded its five-minute timeout. Note that the vertical axis is logarithmic. Clearly, “Old FF” (the original FF) is much too slow for practical use. The same is true – although to a less dramatic extent – for LAMA, which already takes 3 seconds to generate the relatively short sentence at $n = 3$ and times out for larger instances. “New preprocessor” is FF with only our fixes to FF’s preprocess, “New FF” applies all fixes. The preprocessor fixes alone lead to a speed-up of at least 3 orders of magnitude, leading to practically useful runtime – e.g., the 24-word sentence at $n = 4$ is generated in under two seconds. By contrast, the search configuration does not make much of a difference. This changes in our next experiment.

Results for $d = 2$ are shown in Fig. 3 (b). Here the old FF preprocessor crashed with a segmentation fault even at $n = 1$. LAMA performs better, but times out beyond $n > 2$. Best-first search with helpful actions is clearly the best-performing search strategy here. It takes about 14 seconds for $n = 3$, but this sentence is already quite long at 29 words, and EHC takes 3.5 minutes on the same input. Best-first search without helpful actions timed out at $n = 3$, illustrating the importance of that pruning technique.

We remark that the relative behavior of the different search methods is not uniform in our domain, which exhibits a huge amount of structural variance due to the many different forms that the input grammar may take. We are far from an exhaustive exploration of all the possible parameter settings. For instance, with particular inputs (no distractors), EHC can be more effective than best-first search.

Conclusion

Generation of natural language sentences is interesting as a planning problem, both in its application value and in the challenges it poses. After the changes to FF described in this paper, for the first time an automated planner is a practical option for this application. The simplicity of the changes required is striking, and strongly suggests to look more closely at automatic planner configuration techniques.

The domain is far from “solved” in all its aspects. As we pointed out, several features of the domain are very hard to capture with existing search heuristics. In Fig. 3, this shows in the fact that no planner version scales beyond $n > 4$ – even in case (a) where no adjectives at all are required. It remains an interesting challenge to address this.

Acknowledgements. We thank Konstantina Garoufi for driving sentence generation with FF to its limits, Ron Petrick for fruitful discussions, Silvia Richter for making LAMA available, and the reviewers for their helpful comments.

References

Cai, D.; Hoffman, J.; and Helmert, M. 2009. Enhancing the context-enhanced additive heuristic with precedence constraints. In *Proc. ICAPS’09*.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What’s the difference anyway? In *Proc. ICAPS’09*, 162–169.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *Proc. ICAPS’07*, 176–183.

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In *Proc. ICAPS’04*.

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

Hoffmann, J. 2005. Where ‘ignoring delete lists’ works: Local search topology in planning benchmarks. *JAIR* 24:685–758.

Hutter, F.; Hoos, H.; and Stützle, T. 2007. Automatic algorithm configuration based on local search. In *Proc. AAAI ’07*.

Joshi, A., and Schabes, Y. 1997. Tree-Adjoining Grammars. In Rozenberg, G., and Salomaa, A., eds., *Handbook of Formal Languages*, volume 3. Springer.

Karpas, E., and Domshlak, C. 2009. Cost-optimal planning with landmarks. In *Proc. IJCAI’09*.

Koehler, J., and Hoffmann, J. 2000. On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *JAIR* 12:338–386.

Koller, A., and Petrick, R. 2010. Experiences with planning for natural language generation. *CI*. To appear.

Koller, A., and Stone, M. 2007. Sentence generation as planning. In *Proc. of the 45th Annual Meeting of the Association for Computational Linguistics (ACL’07)*.

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

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *Proc. AAAI’08*.

Steedman, M., and Petrick, R. P. A. 2007. Planning dialog actions. In *Proceedings of the Eighth SIGdial Workshop on Discourse and Dialogue*, 265–272.

Stone, M.; Doran, C.; Webber, B.; Bleam, T.; and Palmer, M. 2003. Microplanning with communicative intentions: The SPUD system. *CI* 19(4).

XTAG Research Group. 2001. A lexicalized tree adjoining grammar for english. Technical Report IRCS-01-03, IRCS, University of Pennsylvania.