Using AI Planning Techniques in OpenCCG: Detecting Infeasible Composites in Sentence Generation

Maximilian Schwenger, Álvaro Torralba, Jörg Hoffmann,
David Howcroft, and Vera Demberg

Saarland University,
Saarbrücken, Germany
schwenger@stud.uni-saarland.de, {torralba,hoffmann}@cs.uni-saarland.de,
{howcroft,vera}@coli.uni-saarland.de

Abstract. OpenCCG sentence generation is a prominent approach to *surface realization* – choosing the formulation of a sentence – via search. Such searches often visit many *infeasible* composites (partial sentences), not part of any complete sentence because their grammar category cannot be extended to a sentence in a way covering exactly the desired sentence meaning. Formulating the completion of a composite into a sentence as finding a solution path in a large state-transition system, we exhibit a connection to AI Planning, and we design a compilation from OpenCCG into planning allowing to detect infeasible OpenCCG composites via AI Planning dead-end detection methods. Our experiments show that this can filter out large fractions of infeasible states in, and thus benefit the performance of, complex surface realization processes.

1 Introduction

OpenCCG sentence generation is a prominent approach to surface realization via search [22, 23]. It is based on *combinatory categorial grammars (CCG)*, where words from a lexicon are annotated with syntactic *categories* (e. g. $\mathbf{NP}$ for "noun phrase"), and simple combination rules dictate how categories can be combined. The target of the realization process is a sentence, category $\mathbf{S}$, that conveys the desired meaning, formalized in terms of a set of *semantic items*, where each item must be covered exactly once. The search space (a so-called chart realization algorithm, e. g. [11, 2, 3]) traverses collections of partial sentences (composites, so-called *edges*). One important source of complexity in this context are *infeasible* edges, not part of any complete sentence because their category cannot be extended to a sentence in a way covering exactly the remaining semantic items. Our contribution is a new technique to automatically identify, and prune, infeasible OpenCCG edges, via a connection to AI Planning.

Connections between sentence generation and AI Planning were previously established for so-called tree-adjoining grammars [14, 12, 13], showing how to formulate the entire generation problem as planning. Here we design a new connection for combinatory categorial grammars, and we focus on the objective of identifying infeasible edges, keeping the overall generation process in the hands of OpenCCG. This is better suited for surface realization as a planning compilation would be agnostic of sentence-quality measures, such as n-grams, which are difficult, perhaps impossible, to capture with standard AI Planning quality notions like action costs.

We observe that edge feasibility in OpenCCG – the ability to complete an edge e_0 into a sentence – can be formulated in terms of a state-transition system. We design a compilation of that ability into an AI Planning task Π, where unsolvability of Π – the absence of a path to the planning goal – implies infeasibility of e_0. Applying dead-end detection algorithms from AI Planning (e. g. [6, 9, 8, 19]) to the compiled task Π, and doing so for every edge e_0 during the OpenCCG realization process, then allows to detect and filter out infeasible edges. Our experiments show that this can filter out large fractions of infeasible edges in, and thus benefit the performance of, complex realization processes.

2 Background and State-Transition System Notation

We briefly introduce background and basic notations for OpenCCG and planning, in a manner geared at our compilation techniques.

2.1 OpenCCG

Combinatory categorial grammar (CCG) is a grammar formalism, which, in a nutshell, assigns *(syntactic) categories* to words or sequences thereof, and provides a set of *combination rules* to combine these. Categories can be either atomic, e. g. noun phrase $\mathbf{NP}$, or complex, e. g. $\mathbf{NP}/\mathbf{N}$, where a slash indicates that the sequence $\mathbf{NP}/\mathbf{N}\ \mathbf{N}$ can be combined, via application of the *forward application* rule, to obtain a noun phrase. A backslash requires the combination partner, in *backward application*, to be on the left hand side. As an example, consider the sentence `Winter is coming`, where `Winter` as proper name has category $\mathbf{NP}$, `is` as verb modifier has category $\mathbf{S}\backslash\mathbf{NP}/(\mathbf{S}\backslash\mathbf{NP})$, and `coming` as intransitive verb has category $\mathbf{S}\backslash\mathbf{NP}$. We can combine `is coming` to acquire $\mathbf{S}\backslash\mathbf{NP}$, and a combination thereof with `Winter` results in a sentence, i. e., in category $\mathbf{S}$. There are also unary rules, allowing to change a word sequence's category on its own, to enable different combinations.

In OpenCCG's realization process, a logical formula consisting of a conjunction of elementary predications – the *semantic items* – is transformed into a sentence covering the semantic items. In this process, a *lexicon* provides entries – words associated with categories – potentially useful in terms of their semantics. Composed entries, enriched with additional information, are called *edges* during the search.[1]

Towards our compilation, we next give notations for OpenCCG, and OpenCCG realization, already following AI Planning terminology. In doing so, we will not keep track of the word sequences in edges, and we will not incorporate any notion of word-sequence quality. This is because the purpose of our work merely is to filter out infeasible edges. We specify only those aspects relevant to that purpose.

We refer to the input of the realization process as an *OpenCCG task*, notated $\Omega = (C_0^\Omega, SI^\Omega, R^\Omega, s_I^\Omega, e_G^\Omega)$. Here, C_0^Ω is the finite set of atomic categories c_0. SI^Ω is the

[1] We find the name clash with "edges" in a graph unfortunate, but stick to this terminology here as it is standard in the OpenCCG literature.

finite set of semantic items si. R^{Ω} is the set of combination rules. We will denote the set of *all* categories, that can be formed from C_0^{Ω} through applying the rules R^{Ω}, by C^{Ω}. s_I^{Ω} is what we call a *state*, a set of *edges*, namely those edges already reached in the state. s_I^{Ω} specifically is the *initial state*. An edge e is a pair (c, σ) where $c \in C^{\Omega}$ is a category and $\sigma \subseteq SI^{\Omega}$ is the subset of semantic items *covered* by e (which we will also refer to as the edge's *coverage*). We denote the set of all edges by E^{Ω}. The initial state $s_I^{\Omega} \subseteq E^{\Omega}$ corresponds to the words in the lexicon. Finally, e_G^{Ω} is the *goal edge*, defined as $e_G^{\Omega} = (\mathbf{S}, SI^{\Omega})$.

Given this input, OpenCCG realization conducts a search – a chart realization process – over the possible constructions of new edges from previous ones. Each step of the search either applies a unary rule to an edge already reached, i. e., an edge contained in the current state; or applies a combination rule to a pair of edges $e_1 = (c_1, \sigma_1)$ and $e_2 = (c_2, \sigma_2)$ from the current state, where c_1 and c_2 can be combined, and the truth value assignments have *empty overlap*, $\sigma_1 \cap \sigma_2 = \emptyset$. The resulting new edge is added into the outcome state. The details of how this search process is organized are not relevant to our purpose. Relevant to us are the states and their transitions, which we notate as Ω's *OpenCCG state space*, $\Theta^{\Omega} = (S^{\Omega}, T^{\Omega}, s_I^{\Omega}, S_G^{\Omega})$. Here, $S^{\Omega} = \mathcal{P}(E^{\Omega})$ is the set of all possible states; $T^{\Omega} \subseteq S^{\Omega} \times S^{\Omega}$ contains the transitions over states, as just explained; s_I^{Ω} is the initial state; and $S_G^{\Omega} := \{s_G^{\Omega} \in S^{\Omega} \mid e_G^{\Omega} \in s_G^{\Omega}\}$, the *goal states*, are those containing e_G^{Ω}. We say that a state s^{Ω} is *reachable* in Θ^{Ω} if there is a transition path from s_I^{Ω} to s^{Ω} in Θ^{Ω}. The reachable states in Θ^{Ω} correspond to the OpenCCG search space. We say that Ω is *solvable* if Θ^{Ω} contains a reachable goal state.

Formulating our example above in this manner, say the semantic items SI^{Ω} are $\{\text{Winter}, \text{be}, \text{comes}\}$. Say the lexicon contains exactly the three words needed, so that the initial state s_I^{Ω} contains the edges $(\mathbf{NP}, \{\text{Winter}\})$, $(\mathbf{S}\backslash\mathbf{NP}/(\mathbf{S}\backslash\mathbf{NP}), \{\text{be}\})$, and $(\mathbf{S}\backslash\mathbf{NP}, \{\text{come}\})$. A solution to Θ^{Ω} then is the path $s_I^{\Omega} \to s_1^{\Omega} \to s_2^{\Omega}$ where $s_1^{\Omega} = s_I^{\Omega} \cup \{(\mathbf{S}\backslash\mathbf{NP}, \{\text{be}, \text{come}\})\}$ and $s_2^{\Omega} = s_1^{\Omega} \cup \{(\mathbf{S}, \{\text{Winter}, \text{be}, \text{come}\})\}$.

We say that an edge e_0 is *feasible* in an OpenCCG task Ω iff, in the OpenCCG state space Θ^{Ω}, there is a reachable goal state $s_G^{\Omega} \in S_G^{\Omega}$ *containing a derived tree* T_0 for e_G^{Ω} where e_0 appears in T_0. Here, the derived tree T for an edge e is a tree combining edges to get from elements of s_I^{Ω} to e; and a state s^{Ω} contains T if all edges in T are elements of s^{Ω}. In other words, e_0 is feasible if it forms part of a derived tree for a complete sentence. Otherwise, e_0 is *infeasible*.

2.2 AI Planning

We consider *STRIPS Planning* [5], over Boolean variables (facts), extended with *conditional effects* [17]. This has wide-spread support in modern planning techniques, and matches the needs of our desired OpenCCG compilation.

A *planning task* is a tuple $\Pi = (F^{\Pi}, A^{\Pi}, s_I^{\Pi}, G^{\Pi})$. Here, F^{Π} is a finite set of *facts*; $s_I^{\Pi} \subseteq F^{\Pi}$ is the *initial state* (the facts initially true); and G^{Π} is the *goal* (the facts we need to be true at the end). A^{Π} is a finite set of *actions*. Each action $a \in A^{\Pi}$ is a tuple $(pre_a, add_a, del_a, CEff_a)$ where $pre_a \subseteq F^{\Pi}$ is the action's *precondition*, $add_a \subseteq F^{\Pi}$ is the action's *add list*, $del_a \subseteq F^{\Pi}$ is the action's *delete list*, and $CEff_a$ is the action's finite set of *conditional effects*. Each $e \in CEff_a$ is a triple (con_e, add_e, del_e) of fact sets, namely the effect's *condition*, *add list*, and *delete list* respectively.

Given a planning task Π, the task's *state space* is a tuple $\Theta^{\Pi} = (S^{\Pi}, T^{\Pi}, s_I^{\Pi}, S_G^{\Pi})$. Here, $S^{\Pi} = \mathcal{P}(F^{\Pi})$ is the set of all possible states, i. e., fact subsets interpreted as those facts currently true; s_I^{Π} is Π's initial state; and $S_G^{\Pi} := \{s_G \in S^{\Pi} \mid G^{\Pi} \subseteq s_G\}$ are the *goal states*, where Π's goal is true. The state transitions $T^{\Pi} \subseteq S^{\Pi} \times S^{\Pi}$ arise from action applications. Action a is *applicable* in state s if $pre_a \subseteq s$; in that case, the outcome state is defined as $s' := (s \cup add_a \cup \bigcup_{e \in CEff_a : con_a \subseteq s} add_e) \setminus (del_a \cup \bigcup_{e \in CEff_a : con_a \subseteq s} del_e)$. In other words, s' results from s by including the add lists of the action plus those effects whose condition holds in s, and afterwards removes the delete lists of the action and those effects.[2] We say that Π is *solvable* if Θ^{Π} contains a reachable goal state.

3 Partial Compilation of OpenCCG Sentence Generation into AI Planning

There is a correspondence between AI Planning and OpenCCG realization – at the level of category combination rules and semantic item coverage – in that both require to reach a goal, from an initial state, in a transition system described in terms of actions/transition rules. We aim at exploiting this connection, via a *compilation* from OpenCCG into AI Planning, for automatic filtering of infeasible edges.

Our compilation is *partial* in that it does not attempt to preserve OpenCCG edge reachability exactly. The compilation makes approximations – losing information – aimed at practically viability. It consists of (1) a finite approximation of the set of reachable categories; (2) a planning task capturing solvability of Ω, modulo approximation (1) plus an approximation of semantic coverage; (3) a modified planning task capturing edge feasibility. We introduce these constructions in this order.

3.1 Finite Approximations of Reachable Categories

In CCG, combination rules specify how to create new categories from old ones. It is possible in principle to simulate this behavior in terms of AI Planning actions, designed to emulate the behavior of CCG combination rules. But this yields large and complex planning encodings, and it is not clear how to exploit those effectively. Therefore, here we take a different approach, pre-compiling all combined (non-atomic) categories that will be considered by the planning process. Our starting point is what we call the *category space*, capturing all possible categories and compositions:

Definition 1. *Let $\Omega = (C_0^{\Omega}, SI^{\Omega}, R^{\Omega}, s_I^{\Omega}, e_G^{\Omega})$ be an OpenCCG task. The* category space *of Ω is the pair $(C^{\Omega}, \gamma^{\Omega})$ where $\gamma^{\Omega} : C^{\Omega} \times C^{\Omega} \cup C^{\Omega} \mapsto \mathcal{P}(C^{\Omega})$ is the partial function where $c' \in \gamma^{\Omega}(c)$ iff c can be transformed into c' using a unary rule from R^{Ω}, and $c' \in \gamma^{\Omega}(c_1, c_2)$ iff c_1 and c_2 can be combined into c' using a binary rule from R^{Ω}.*

Note that γ^{Ω} is a function onto subsets of possible outcome categories, rather than onto a unique outcome category, as several different rules may be applicable to the same

[2] For some purposes, one needs to design a special treatment for conflicting effects, adding vs. deleting the same fact. This will not be relevant in our context.

input categories. Note further that, in the presence of unary rules (like type raising) which are always applicable in CCG, the category space is infinite. To compile it into a finite planning task, we need to restrict ourselves to a finite sub-space. We do so via a size-bound parameter k, in an optimistic vs. a pessimistic manner:

Definition 2. *Let $\Omega = (C_0^\Omega, SI^\Omega, R^\Omega, s_I^\Omega, e_G^\Omega)$ be an OpenCCG task. Let k be a natural number. For $c \in C^\Omega$, let the* degree *of c, denoted $\#(c)$, be the overall number of slashes and backslashes in c. By $C^\Omega[k] := \{c \in C^\Omega \mid \#(c) \leq k\} \cup \{*\}$, we denote the set of all categories whose degree is at most k, plus the* wildcard *symbol $*$.*

The pessimistic category space *of Ω given k is the pair $(C^\Omega[k], \gamma^{-\Omega})$ where $\gamma^{-\Omega}$ is defined like γ^Ω but replacing any category c' where $\#(c') > k$ by $*$. The* optimistic category space *of Ω given k is the pair $(C^\Omega[k], \gamma^{+\Omega})$ where $\gamma^{+\Omega}$ is defined like γ^Ω but replacing any category c' where $\#(c') > k$ by $*$; and including $c' \in \gamma^\Omega(*)$ whenever $\gamma^\Omega(c) = c'$; and including $c' \in \gamma^\Omega(c_1, *)$ whenever $\gamma^\Omega(c_1, c_2) = c'$; and including $c' \in \gamma^\Omega(*, c_2)$ whenever $\gamma^\Omega(c_1, c_2) = c'$.*

In other words, we cut off the generation of categories once their degree exceeds a user-defined threshold k. In the pessimistic (under-approximating) variant, no further combinations are possible behind $*$. In the optimistic (over-approximating) variant, all combinations are possible behind $*$.[3]

Say that, in our example, the lexicon contains only the words $(\mathbf{S}\backslash\mathbf{NP}, \{\text{come}\})$ and $(\mathbf{S}\backslash\mathbf{NP}/(\mathbf{S}\backslash\mathbf{NP}), \{\text{be}\})$. If we set $k := 3$, then $\gamma^{+\Omega}$ preserves γ^Ω sufficiently to determine that $\mathbf{S}$ cannot be reached from the initial state categories. For $k := 2$, however, $\mathbf{S}\backslash\mathbf{NP}/(\mathbf{S}\backslash\mathbf{NP})$ is replaced by $*$, and we can reach $\mathbf{S}$ by "pretending" that $*$ stands for $\mathbf{NP}$.

The optimistic approximation variant preserves solutions and can be used to provide guarantees, i. e., using our edge-feasibility compilation below, to prune only edges that are indeed infeasible. The pessimistic variant does not provide that guarantee, but tends to be more successful in practice as we will show in Section 5. Observe that the approximations approach γ^Ω from opposite sides, in the sense that they are coarsest for $k = 1$, and become more precise as k grows, $\gamma^{+\Omega}$ getting less optimistic and $\gamma^{-\Omega}$ getting less pessimistic. The approximations converge to γ^Ω in that, for any finite sub-space of $(C^\Omega, \gamma^\Omega)$, there is a k so that both approximations are exact. In terms of edge pruning, this means that the optimistic variant prunes more for larger k, and eventually is precise enough to find any edge that can be pruned; while the pessimistic variant prunes less for larger k, and eventually is precise enough to preserve any edge that cannot be pruned (in particular: precise enough to preserve any one solution).

3.2 Planning Compilation for Solvability

To capture solvability relative to the optimistic/pessimistic finite category space approximation, our compiled planning task combines facts keeping track of category creation

[3] One can (and our implementation does) define $\gamma^{+\Omega}$ in a more fine-grained manner, replacing only the *sub*-categories behind the threshold k with $*$, and accordingly being less generous in the over-approximation of γ. As this refined version is cumbersome to spell out formally, and leads to similar results in practice, we omit this here.

with facts keeping track of semantic coverage. Here again we face a design choice: we could, in principle, keep track of the actual category/coverage pairs, i. e., of edges. This would allow us to check for empty overlap when combining two edges. However, that would (a) again yield rather large planning encodings, and (b) require the AI Planning dead-end detection method to be able to reason about delete lists. The most canonical dead-end detection method, that we employ here, does not qualify for (b), so we can just as well circumvent (a). We do so by abstracting from edges, associating a category c with a semantic item si if *at least one* reached edge has category c and covers si. We compile this into a planning task as follows:

Definition 3. *Let $\Omega = (C_0^\Omega, SI^\Omega, R^\Omega, s_I^\Omega, e_G^\Omega)$ be an OpenCCG task. Let k be a natural number. The* optimistic solvability-compilation *is the planning task $\Pi^{+\Omega}[k] = (F^\Pi, A^\Pi, s_I^\Pi, G^\Pi)$ where:*

(i) $F^\Pi = \{c \mid c \in C^\Omega[k]\} \cup \{c[si] \mid c \in C^\Omega[k], si \in SI^\Omega\}$.
(ii) $s_I^\Pi = \{c \mid e = (c, \sigma) \in s_I^\Omega\} \cup \{c[si] \mid e = (c, \sigma) \in s_I^\Omega, si \in \sigma\}$.
(iii) $G^\Pi = \{\mathbf{S}\} \cup \{\mathbf{S}[si] \mid si \in SI^\Omega\}$.
(iv) $A^\Pi = \{a[c, c'] \mid \gamma^{+\Omega}(c) = c'\} \cup \{a[c_1, c_2, c'] \mid \gamma^{+\Omega}(c_1, c_2) = c'\}$, *where:*
  (a) $a[c, c'] := (\{c\}, \{c'\}, \emptyset, \{(\{c[si]\}, \{c'[si]\}, \emptyset) \mid si \in SI^\Omega\})$.
  (b) $a[c_1, c_2, c'] := (\{c_1, c_2\}, \{c'\}, \emptyset, \{(\{c_j[si]\}, \{c'[si]\}, \emptyset) \mid j \in \{1, 2\}, si \in SI^\Omega\})$.

The pessimistic solvability-compilation *is the planning task $\Pi^{-\Omega}[k]$, defined like $\Pi^{+\Omega}[k]$ but using $\gamma^{-\Omega}$.*

Items (i)–(iii) should be easy to understand: in the compiled planning task, facts c indicate whether category c has been reached yet, and facts $c[si]$ indicate whether c covers si yet; in the initial state, these flags are set according to s_I^Ω, i. e., according to the words in the lexicon; the goal is to have a sentence covering the entire semantics. To understand item (iv), recall that actions have the form $(pre_a, add_a, del_a, CEff_a)$. In item (iv a), encoding unary rule applications $\gamma^{+\Omega}(c) = c'$, the precondition is $\{c\}$ and the (unconditional) add list is $\{c'\}$, effectively saying that, if c is already reached, then applying the action (the rule) yields c'. The conditional effects simply transfer, for each si, the coverage from c (if already reached) to c'. The encoding of binary rules in item (iv b) is similar.

Note that, as indicated above, the delete lists in the compilation are empty. On the one hand, this corresponds to the monotonic nature of the OpenCCG search space, where new edges are being added without removing the old ones. On the other hand, delete effects *would* be needed to capture empty coverage overlap in combination-rule applications. Yet, as explained, in the present approach we forsake that information as our dead-end detector would not be able to handle it anyhow.

Theorem 1. *Let Ω be an OpenCCG task. Let k be a natural number. If Ω is solvable, then so is $\Pi^{+\Omega}[k]$.*

Proof. The proof compares Ω's state space $\Theta^\Omega = (S^\Omega, T^\Omega, s_I^\Omega, S_G^\Omega)$ with that of $\Pi^{+\Omega}[k]$. Denote the latter by $\Theta = (S, T, s_I, S_G)$. Define the mapping $\alpha : S^\Omega \mapsto S$ as $\alpha(s^\Omega) := \{c \mid e = (c, \sigma) \in s^\Omega\} \cup \{c[si] \mid e = (c, \sigma) \in s^\Omega, si \in \sigma\}$. As $\alpha^{+\Omega}$ over-approximates the category combinations in α^Ω, it is easy to see that transitions are preserved by α, i. e., whenever $(s_1^\Omega, s_2^\Omega) \in T^\Omega$, we have $(\alpha(s_1^\Omega), \alpha(s_2^\Omega)) \in T$. Furthermore, goal states are preserved, i. e., whenever $s^\Omega \in S_G^\Omega$, we have $\alpha(s^\Omega) \in S_G$. Finally, $\alpha(s_I^\Omega) = s_I$. The claim follows.

Given Theorem 1, if $\Pi^{+\Omega}[k]$ is *not* solvable, i. e., if an AI Planning dead-end detector is able to detect that this is so, then we can safely conclude that Ω is not solvable either. The pessimistic compilation $\Pi^{-\Omega}[k]$ does not give that guarantee.

For illustration, consider again the example variant where the lexicon contains only the words $(\mathbf{S}\backslash\mathbf{NP}/(\mathbf{S}\backslash\mathbf{NP}), \{\text{be}\})$ and $(\mathbf{S}\backslash\mathbf{NP}, \{\text{come}\})$, so Ω is unsolvable. Then $\Pi^{+\Omega}[3]$ is unsolvable as $\mathbf{S}$ cannot be reached using $\gamma^{+\Omega}$, cf. above. $\Pi^{+\Omega}[2]$ also is unsolvable, because the semantic item "Winter" cannot be covered; but if we remove that semantic item from the OpenCCG task (and thus from $\Pi^{+\Omega}[2]$), then $\Pi^{+\Omega}[2]$ has a one-step solution combining the two initial-state categories.

3.3 Planning Compilation for Edge Feasibility

The above compilation provides a necessary criterion for an OpenCCG task to be solvable. However, our actual purpose requires a necessary criterion for an OpenCCG *edge* e_0 to be *feasible*, i. e., to form part of a solution. This can be achieved by a simple modification of the compilation, propagating markers to make sure that e_0's category is used in the solution:[4]

Definition 4. *Let* $\Omega = (C_0^\Omega, SI^\Omega, R^\Omega, s_I^\Omega, e_G^\Omega)$ *be an OpenCCG task, and let* $e_0 = (c_0, \sigma_0)$ *be an edge in* Ω*. Let k be a natural number. The* optimistic feasibility-compilation *is the planning task* $\Pi^{+\Omega}[k, e_0] = (F^\Pi, A^\Pi, s_I^\Pi, G^\Pi)$ *where:*

(i) $F^\Pi = \{c, c[0] \mid c \in C^\Omega[k]\} \cup \{c[si] \mid c \in C^\Omega[k], si \in SI^\Omega\}$.
(ii) $s_I^\Pi = \{c_0[0]\} \cup \{c \mid e = (c,\sigma) \in s_I^\Omega, \sigma \cap \sigma_0 = \emptyset\} \cup \{c[si] \mid e = (c,\sigma) \in s_I^\Omega, \sigma \cap \sigma_0 = \emptyset, si \in \sigma\}$.
(iii) $G^\Pi = \{\mathbf{S}, \mathbf{S}[0]\} \cup \{\mathbf{S}[si] \mid si \in SI^\Omega\}$.
(iv) $A^\Pi = \{a[c,c'] \mid \gamma^{+\Omega}(c) = c'\} \cup \{a[c_1,c_2,c'] \mid \gamma^{+\Omega}(c_1,c_2) = c'\}$*, where:*
  (a) $a[c,c'] := (\{c\}, \{c'\}, \emptyset, \{(\{c[0]\}, \{c'[0]\}, \emptyset)\} \cup \{(\{c[si]\}, \{c'[si]\}, \emptyset) \mid si \in SI^\Omega\})$.
  (b) $a[c_1,c_2,c'] := (\{c_1,c_2\}, \{c'\}, \emptyset, \{(\{c_1[0]\}, \{c'[0]\}, \emptyset), (\{c_2[0]\}, \{c'[0]\}, \emptyset)\} \cup \{(\{c_j[si]\}, \{c'[si]\}, \emptyset) \mid j \in \{1,2\}, si \in SI^\Omega\})$.

The pessimistic feasibility-compilation *is* $\Pi^{-\Omega}[k, e_0]$*, defined like* $\Pi^{+\Omega}[k, e_0]$ *but using* $\gamma^{-\Omega}$*.*

Relative to Definition 3, we add the $c[0]$ markers to keep track of whether an ancestor of c uses e_0's category. The initial state includes this marker only for e_0's own category c_0, the goal is for $\mathbf{S}$ to be marked. The actions propagate the markers through conditional effects, marking the outcome category c' if at least one of the input categories is already marked. The additions "$\sigma \cap \sigma_0 = \emptyset$" in (ii) introduce a limited form of empty coverage overlap reasoning, excluding in the initial state those edges whose semantics overlaps with e_0 (and that thus won't be used in a solution incorporating e_0).

[4] In this definition, the modifications relative to Definition 3 are shown in red for the benefit of on-screen reading.

Theorem 2. *Let Ω be an OpenCCG task, and let e_0 be an edge in Ω. Let k be a natural number. If e_0 is feasible in Ω, then $\Pi^{+\Omega}[k, e_0]$ is solvable.*

Proof. Say that e_0 is feasible in Ω. Then there is a solution θ to Ω using e_0, and not using any $e \in s_I^{\Omega}$ whose semantics overlaps with that of e_0. By Theorem 1, $\Pi^{+\Omega}[k]$ is solvable, via a transition path π corresponding to θ. By construction, π is a solution for $\Pi^{+\Omega}[k, e_0]$.

Given Theorem 2, if an AI Planning dead-end detector proves $\Pi^{+\Omega}[k, e_0]$ to be unsolvable, then we can conclude that e_0 is infeasible. The pessimistic compilation $\Pi^{-\Omega}[k, e_0]$ does not give that guarantee.

Say that, in our example, the lexicon contains the words $e_1 = (\mathbf{NP}, \{\text{Winter}\})$, $e_2 = (\mathbf{S}\backslash\mathbf{NP}/(\mathbf{S}\backslash\mathbf{NP}), \{\text{be}\})$, and $e_3 = (\mathbf{S}\backslash\mathbf{NP}, \{\text{come}\})$ as before, but contains also the transitive form of "coming", $e_4 = ((\mathbf{S}\backslash\mathbf{NP})/\mathbf{NP}, \{\text{come}\})$, infeasible for our purposes. Consider the edge $e_0 = (\mathbf{S}\backslash\mathbf{NP}, \{\text{Winter}, \text{come}\})$, where we combined e_1 with e_4. Consider the compilation $\Pi^{+\Omega}[3, e_0]$: In the initial state, as e_1 overlaps e_0, e_1 is not included. But without $\mathbf{NP}$, $\mathbf{S}$ is unreachable given $\gamma^{+\Omega}$ for $k = 3$, so $\Pi^{+\Omega}[3, e_0]$ is unsolvable and we correctly detect that e_0 is infeasible.[5]

4 Practical Compilation Use and Optimizations

Our idea is to create, and check the solvability of, the compiled planning task $\Pi^{+\Omega}[k, e_0]$ respectively $\Pi^{-\Omega}[k, e_0]$, every time a new edge e_0 is created during the OpenCCG realization process. If the compiled planning task is unsolvable, e_0 is deemed infeasible, and is discarded. This filtering method is provably sound when using $\Pi^{+\Omega}[k, e_0]$. When using $\Pi^{-\Omega}[k, e_0]$, it is a practical heuristic, and converges to sound pruning – eventually preserving the best solution – as k grows.

To realize this approach, we require a method for checking solvability of planning tasks. In general, however, deciding solvability ("plan existence") is **PSPACE**-complete [1]. For fast solvability detection, planning research therefore concentrates on polynomial-time solvable fragments of the plan existence problem. The most widespread such fragment is the one where all delete lists are required to be empty (e. g. [1, 6, 9]). Hence the design of our compilation, which incorporates approximations resulting in empty delete lists. For planning tasks with empty delete lists, plan existence can be decided in time low-order polynomial in the size of the task, using so-called *relaxed planning graphs* [9].

Though polynomial time, testing delete-free plan existence does incur a runtime overhead, especially in our context where we need to do so for every edge during realization. Efficient implementation is therefore important. One key to this is the re-use of information/computation shared across individual tests. First, every call to sentence realization based on the same lexicon shares the same category space. Hence we can

[5] Note that the same is not true for $k = 2$; and neither for e_4 because, there, ignoring overlap in rule applications means that e_1 could be used twice. These are weaknesses of our current approach, which could potentially be tackled by more informed compilations. We get back to this in the conclusion.

build the category space approximation, $(C^{\Omega}[k], \gamma^{+\Omega})$ respectively $(C^{\Omega}[k], \gamma^{-\Omega})$, *offline*, just once for the lexicon at hand, prior to realization. Second, the feasibility compilations for individual edges e_0 during the same realization process are identical except for their initial states. So, during a realization process, we create a compiled task just once and adapt it minimally for each test.

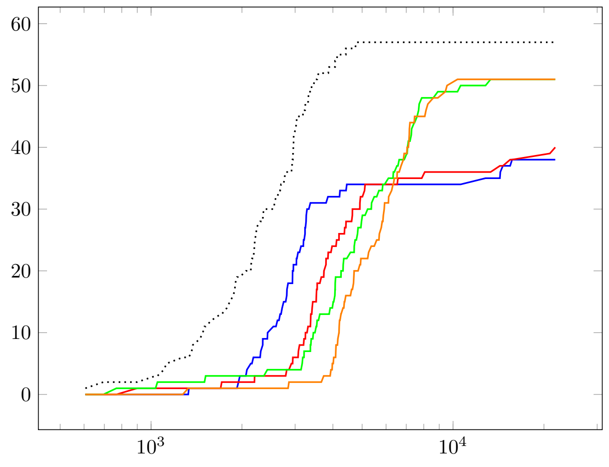
Finally, (a) the action set in $\Pi^{+\Omega}[k, e_0]$ respectively $\Pi^{-\Omega}[k, e_0]$ is fully determined by $\gamma^{+\Omega}$ respectively $\gamma^{-\Omega}$ along with the set of semantic items SI^{Ω}; while (b) for the maintenance of semantic coverage and $c[0]$ markers, instead of the compilation via conditional effects as specified, one can implement a simple special-case handling in the standard relaxed planning graph solvability test. Taking these two observations together, we can generate the action set completely offline. Online, prior to a realization process, we merely need to read in the actions and setup the marker-maintenance data structures.

5 Experiments

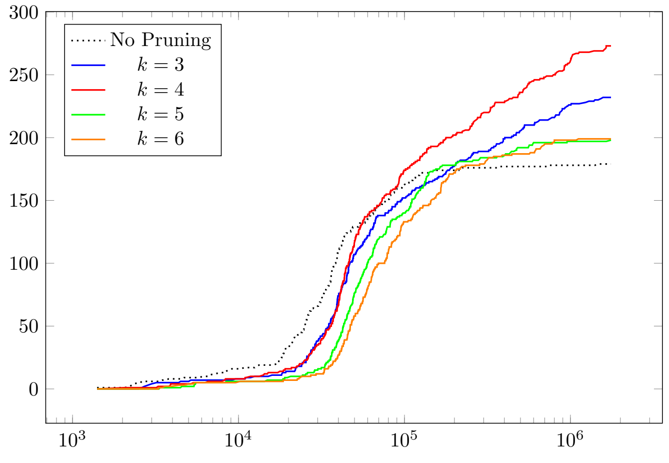
As our main test base for experimentation, we used the SPaRKy Restaurant Corpus (we also ran preliminary experiments with some other test bases, which we get back to below). SPaRKy is one of the only published resources for NLG which provides intermediate representations in addition to system inputs and outputs, and quality ratings for those outputs. Originally introduced by Walker et al. [21], Nakatsu and White [16] developed a CCG grammar for this dataset which spans both the sentence and the discourse levels. The domain of the corpus is restaurant descriptions, including prices, kind of food, decor, service, etc. In this work we use the a set of 431 test instances developed for the contrast-enhanced version of the grammar presented in Howcroft, Nakatsu, & White [10]. The lexicon in this testbed includes 193 words and the grammar is capable of producing a wide variety of texts of varying lengths. Of the 431 OpenCCG realization tasks, 61 *recommendation tasks* require generating a text recommending a single restaurant, while 370 *comparison tasks* require generating a text comparing two or more restaurants with each other. As these instances correspond to the generation of entire text paragraphs, they are complex enough to be interesting use cases for our techniques. This pertains in particular to the comparison tasks, where the required text is longer.

In preliminary tests with the optimistic approximation variant, the pruning was too weak to pay off, i. e., too few edges were pruned to get a benefit. We therefore concentrate here on pruning with the pessimistic approximation variant, where small values of k may prune too aggressively, while large values of k yield more reliable pruning yet incur a larger runtime overhead. All experiments were run on a cluster of machines with Intel Xeon E5-2660 processors running at 2.2 GHz. The runtime/memory limit was set to 30 minutes/4 GB for each sentence generation task, i. e., for each benchmark instance.

As a simple measure of performance, we focus on the runtime spent by the OpenCCG chart realization process until the first solution – the first edge of category $\mathbf{S}$ covering all semantic items – is generated. Figure 1 shows coverage, i. e., the number of benchmark instances where a solution was found, as a function of runtime. We distinguish between



(a)



(b)

Fig. 1. Coverage, i. e., the number of sentence generation tasks to which a solution was found, as a function of runtime on SPaRKy (a) recommendation tasks and (b) comparison tasks. A data point (x, y) means that y tasks are solved within a time limit of x milliseconds. "No pruning" is the baseline OpenCCG search without pruning; "$k = n$" considers our pessimistic pruning with parameter k, i. e., the $\Pi^{-\Omega}[k, e_0]$ compilation, on every edge e_0 during search, pruning e_0 if $\Pi^{-\Omega}[k, e_0]$ is unsolvable.

(a) recommendation vs. (b) comparison tasks as these are different in nature and yield very different performance profiles.

Regarding (a), we see that these tasks are essentially too easy for our pruning method to pay off: the runtime overhead of repeatedly checking the solvability of $\Pi^{-\Omega}[k, e_0]$ outweighs the gain from pruning, so that fewer tasks are solved within the same runtime limits. The restaurant comparison tasks (b), however, are more challenging (notice the different x-axis scales in (a) and (b)), and the picture is different: in the much larger search spaces, the pruning impact is stronger. For runtime limits > 56 seconds, the coverage of $k = 4$ pruning exceeds that without pruning. For runtime limits > 206 seconds, all settings of k exceed the baseline. Note here that the value of k controls the trade-off between accuracy (better with large k) and runtime overhead (better with small k). In SPaRKy restaurant comparison tasks, $k = 4$ is the sweet spot of that trade off. At our maximum time limit of $x = 30$ minutes, $k = 4$ pruning increases coverage from 179 instances without pruning, to 273 with pruning, an increase of 52%.

As the pessimistic compilation does not guarantee that pruned edges are actually infeasible, the pruning may adversely affect the quality of the sentences generated. As k gets larger, this danger decreases as the pruning becomes more accurate. In SPaRKy, it turns out that $k = 4$ is not only best in terms of runtime performance, but is also enough to avoid any deterioration in sentence quality. Of the 215 instances solved by both the baseline and $k = 4$ (across recommendation and comparison tasks), in 137 cases the two realizations are identical. In the remainining 78 cases, the realizations differ only in using the word “just” vs. the word “only”, so that the version with pruning does exactly as well as the baseline.

Going beyond solutions, there also are cases where OpenCCG produces a *partial solution*, an edge of category $\mathbf{S}$ that covers only a subset of the semantic items. This can still be useful if, e. g., four instead of five restaurants are being compared. Our $k = 4$ pruning has clear advantages in terms of the ability to find such partial solutions. Of the 97 cases where neither $k = 4$ nor the baseline find a complete solution, $k = 4$ provides a partial solution in 81 cases, the baseline in 52 cases. In all 21 cases where only the baseline finds a complete solution, $k = 4$ finds a partial solution. In contrast, of the 98 cases where only $k = 4$ finds a complete solution, in 59 cases the baseline does not manage to find a partial solution.

We also ran experiments on some other test bases [20, 18, 15], yet as the text paragraphs to be generated were comparatively small, similarly to SPaRKy recommendation tasks our pruning methods generally did not pay off. Conversely, in the CCGBank [7], our category space approximations consumed excessive amounts of memory. For practical viability in such large bases, either additional implementation tricks, or more intelligent abstractions (not just enumerating all categories up to a fixed degree), would be required.

6 Conclusion

Sentence generation as search relates deeply to AI Planning in that, at least as far as grammatical and semantical correctness is concerned, it is essentially a reachability problem in a large discrete transition system. This connection has been made before, and we herein propose a new variant and application, detecting infeasible edges in OpenCCG. Our empirical results show promise, though much remains to be done.

In our view, the most pressing and interesting question is *how much information we can efficiently capture and exploit* in this kind of compilation. Our present approach is (a) inflexible in precomputing a category space approximation, and (b) conservative in targeting delete-free planning which is easy to handle. Both design choices sacrifice information, and both may be lifted through more intelligent compilations/abstractions, paired with more advanced dead-end detection on the AI Planning side, as exhibited e. g. in the inaugural unsolvability-detection planning competition `http://unsolve-ipc.eng.unimelb.edu.au/`.

From a broader point of view, we believe that AI Planning, and AI search techniques more generally, can be a crucial piece in the puzzle for achieving practical sentence generation with complex optimization objectives [4].

References

1. Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1–2):165–204, 1994.
2. Aoife Cahill and Josef van Genabith. Robust pcfg-based generation using automatically acquired LFG approximations. In Nicoletta Calzolari, Claire Cardie, and Pierre Isabelle, editors, *Proceedings of the 21st International Conference on Computational Linguistics (ACL'06)*. ACL, 2006.
3. John A. Carroll and Stephan Oepen. High efficiency realization for a wide-coverage unification grammar. In *Natural Language Processing–IJCNLP*, pages 165–176, 2005.
4. Vera Demberg, Jörg Hoffmann, David Howcroft, Dietrich Klakow, and Alvaro Torralba. Search challenges in natural language generation with complex optimization objectives. *KI – Künstliche Intelligenz*, 30:63–69, 2016.
5. Richard E. Fikes and Nils Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.
6. Patrik Haslum and Hector Geffner. Admissible heuristics for optimal planning. In S. Chien, R. Kambhampati, and C. Knoblock, editors, *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS'00)*, pages 140–149, Breckenridge, CO, 2000. AAAI Press, Menlo Park.
7. Julia Hockenmaier and Mark Steedman. Ccgbank: A corpus of ccg derivations and dependency structures extracted from the penn treebank. *Computational Linguistics*, 33(3):355–396, September 2007.
8. Jörg Hoffmann, Peter Kissmann, and Álvaro Torralba. "Distance"? Who Cares? Tailoring merge-and-shrink heuristics to detect unsolvability. In Thorsten Schaub, editor, *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI'14)*, Prague, Czech Republic, August 2014. IOS Press.
9. Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.
10. David Howcroft, Crystal Nakatsu, and Michael White. Enhancing the expression of contrast in the sparky restaurant corpus. In *Proceedings of the 14th European Workshop on Natural Language Generation (ENLG'13)*, pages 30–39, 2013.
11. Martin Kay. Chart generation. In Aravind K. Joshi and Martha Palmer, editors, *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 200–204. Morgan Kaufmann / ACL, 1996.

12. Alexander Koller and Jörg Hoffmann. Waking up a sleeping rabbit: On natural-language sentence generation with ff. In Ronen I. Brafman, Hector Geffner, Jörg Hoffmann, and Henry A. Kautz, editors, *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10)*. AAAI Press, 2010.
13. Alexander Koller and Ronald Petrick. Experiences with planning for natural language generation. *Computational Intelligence*, 27(1):23–40, 2011.
14. Alexander Koller and Matthew Stone. Sentence generation as planning. In *Proc. of the 45th Annual Meeting of the Association for Computational Linguistics (ACL'07)*, 2007.
15. Geert-Jan M Kruijff, Pierre Lison, Trevor Benjamin, Henrik Jacobsson, Hendrik Zender, Ivana Kruijff-Korbayová, and Nick Hawes. Situated dialogue processing for human-robot interaction. In *Cognitive Systems*, pages 311–364. Springer, 2010.
16. Crystal Nakatsu and Michael White. Generating with discourse combinatory categorial grammar. *Linguistic Issues in Language Technology*, 4(1), 2010.
17. Edwin P.D. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In R. Brachman, H. J. Levesque, and R. Reiter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 1st International Conference (KR-89)*, pages 324–331, Toronto, ON, May 1989. Morgan Kaufmann.
18. Stefania Racioppa. Italian ccg grammar for aliz-e. Technical report, DFKI, 2011.
19. Marcel Steinmetz and Jörg Hoffmann. Towards clause-learning state space search: Learning to recognize dead-ends. In Dale Schuurmans and Michael Wellman, editors, *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI'16)*. AAAI Press, February 2016.
20. Jean Vancoppenolle, Eric Tabbert, Gerlof Bouma, and Manfred Stede. A german grammar for generation in openccg. Number 96, pages 145–150, 2011.
21. Marilyn A. Walker, Amanda Stent, François Mairesse, and Rashmi Prasad. Individual and domain adaptation in sentence planning for dialogue. *Journal of Artificial Intelligence Research*, 30:413–456, 2007.
22. Michael White. Efficient realization of coordinate structures in combinatory categorial grammar. *Research on Language and Computation*, 4(1):39–75, 2006.
23. Michael White and Rajakrishnan Rajkumar. Minimal dependency length in realization ranking. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 244–255, 2012.