# Partial Grounding in Planning using Small Language Models

**Felipe Areces[1], Benjamin Ocampo[2], Carlos Areces[3,4], Martín Domínguez[3], Daniel Gnad[5]**

[1]Stanford University, USA
[2]Université Côte d'Azure, France
[3]Universidad Nacional de Córdoba, Argentina
[4]CONICET, Argentina
[5]Linköping University, Sweden

**Abstract**

The aim of classical automated planning is to find a sequence of actions, a plan, that changes the state of the world from a given initial state to a state that satisfies the goal condition. Most research in the field focuses on heuristic search, which attempts to find a plan on a fully grounded model of the planning task. However, obtaining the full grounding is often infeasible as its size can be exponentially larger than the original input. We follow up on previous work that introduced partial grounding for planning using a relevance prediction estimate obtained from classical machine learning models. These models are trained offline, on a per-domain basis, to estimate how likely it is for a plan to include a given action.

In this article we leverage recent advances in the field of language models in natural language processing (NLP) to improve these estimates. We use small language models to create word embeddings for actions and facts directly from their textual representation. These models provide fixed-length representations for actions and facts reached along a delete-relaxed solution of a planning task, which can be obtained efficiently. We show that these feature vectors can be used to train predictors of action relevance, that consistently identify relevant actions on an established set of hard-to-ground planning benchmarks.

## 1 Introduction

In classical (satisficing) planning, given a model of the environment, usually represented as a set of possible actions schemas and an initial state (described as a set of facts), we aim to find a finite sequence of operators — which are suitable instantiations of the action schemas — that can transform the initial state into a state satisfying the required goal.

Planning models are usually described in the Planning Domain Definition Language (PDDL) (McDermott et al. 1998) in terms of predicates and action schemas with arguments that can be instantiated with a set of objects. Most planners, however, work on a grounded representation without free variables, like STRIPS (Fikes and Nilsson 1971) or FDR (Bäckström and Nebel 1995).

In recent years, much of the research has focused on the *search* aspect of the problem described above and assumes that all relevant instantiations of the action schemas are available. *Grounding* is the process of translating a task in the lifted PDDL representation to an instantiated representation. It requires computing all relevant instantiations that assign objects to the arguments of predicates and action schema parameters, even though only a small fraction of these instantiations might be necessary to solve the task. For many planning domains, the assumption of a fully grounded model cannot be maintained, as the set of all operators might be too large to represent (it often scales exponentially with the size of the lifted representation). Hence, working with a *partially grounded model* can be necessary. Furthermore, search complexity hinges on the number of operators, so that reducing this number (while ensuring plan existence) can directly impact search complexity and improve performance.

The success of planners like FF (Hoffmann and Nebel 2001) or LAMA (Richter, Westphal, and Helmert 2011) in finding plans for many planning tasks is undeniable. However, since most planners use full grounding, they might fail even before search starts whenever a full instantiation is not possible, which makes grounding a crucial limiting factor in the success of satisficing planners. Recently, an alternative approach to solving planning tasks has become more popular, working directly on the lifted PDDL representation (Ridder and Fox 2014; Corrêa et al. 2020; Lauer et al. 2021). Most lifted planners are still based on heuristic search, but struggle to keep up with their grounded counterparts when the search problem is challenging. That is because successor generation is less efficient and availability of strong heuristics for lifted planning is still limited (Corrêa et al. 2021; Wichlacz, Höller, and Hoffmann 2022; Corrêa et al. 2022; Corrêa and Seipp 2022). Besides this, pruning methods that work on PDDL level have been proposed (Sievers et al. 2019; Fišer 2020), but all of these methods preserve completeness, so they are not comparable to our approach.

In this article, we follow up on previous work that introduced *partial grounding for planning* based on relevance prediction using classical machine learning models (Gnad et al. 2019). These models are trained offline on a per-domain basis (using solely the information in the initial and goal description) to estimate how likely it is for a plan to include a given operator. We extend this approach by showing that information present in the set of *relaxed facts*, i.e. the facts achieved along a delete-relaxed solution of the planning task (Hoffmann and Nebel 2001) is also useful to infer operator relevance. Crucially, these facts can be computed very efficiently. In this way, the standard grounding mechanism (that progressively grounds all reachable operators en-

abled from the initial state of the problem) can be adapted to generate operators in the order induced by the predicted relevance score obtained from our machine learning models. More precisely, we leverage recent advances in the field of natural language processing (NLP) to synthesize the information provided by the textual representation of operators and facts into fixed-length feature vectors using word embeddings. We then use these models to encode facts reached along a delete-relaxed solution of a planning task to train a predictor of operator relevance.

We evaluate our approach on several relevant planning domains to empirically demonstrate the potential performance gains, while releasing the set of instances used for training and the obtained relevance prediction models to foster reproducibility and encourage further research on machine learning applications within planning.

We now provide a brief outline of our contributions in this article:

- We show that information obtained from the set of relaxed facts of a planning task can be used to predict operator relevance during grounding.
- We provide a general framework for the evaluation of grounding techniques.
- We release a data set of solved planning tasks for different planning domains, together with all the information required to develop our models.

## 2 Background

In what follows we will mostly assume, for simplicity, that tasks are specified in the STRIPS subset of PDDL (Fikes and Nilsson 1971), but our algorithms and implementation are directly applicable to a larger subset of PDDL containing ADL expressions (Pednault 1989).[1]

A (lifted) PDDL task $\Pi^{PDDL}$ is a tuple $(\mathcal{P}, \mathcal{A}, \Sigma^C, \Sigma^O, I, G)$ where $\mathcal{P}$ is a set of *predicates*, $\mathcal{A}$ is a set of *action schemas*, $\Sigma := \Sigma^C \cup \Sigma^O$ is a non-empty set of *objects* (with constants $\Sigma^C$, and non-constant objects $\Sigma^O$), $I$ is the *initial state*, and $G$ is the *goal* We denote individual parameters of action schemas and predicates with $x, y, z$ and sequences of parameters with $X, Y, Z$. An action schema $a[X]$ is a triple $(\mathsf{pre}(a), \mathsf{add}(a), \mathsf{del}(a))$, consisting of *preconditions*, an *add list*, and a *delete list*, all of which are subsets of $\mathcal{P}$, possibly pre-instantiated with objects from $\Sigma^C$, such that $X$ (the *interface* of $a[X]$) is the set of variables that appear in $\mathsf{pre}(a) \cup \mathsf{add}(a) \cup \mathsf{del}(a)$. $I$ and $G$ are subsets of $\mathcal{P}$, instantiated with objects from $\Sigma$.

A PDDL task $\Pi^{PDDL}$ can be divided into two parts: the domain specification $(\mathcal{P}, \mathcal{A}, \Sigma^C)$ which is common to all instances of the domain, and the task specification $(\Sigma^O, I, G)$ which is different for each instance of a domain.

A (grounded) STRIPS task $\Pi$ is a tuple $(F, O, I, G)$, where $F$ is a set of grounded predicates, called *facts*, and $O$ is a set of grounded action schemas, called *operators*. A *state* $s \subseteq F$ is a set of facts, $I \subseteq F$ is the *initial state* and $G \subseteq F$ is the *goal*. An operator $o$ is *applicable* in a state $s$ if $\mathsf{pre}(o) \subseteq s$. In that case, the outcome state is $s' = (s \setminus \mathsf{del}(o)) \cup \mathsf{add}(o)$, and we write $s \xrightarrow{o} s'$ for the transition from $s$ to $s'$ via $o$. For a sequence of operators $\overline{o}$, we write $s \xrightarrow{\overline{o}} t$ if the operators in $\overline{o}$ can be iteratively applied to $s$, resulting in $t$. A sequence $\overline{o}$, with $I \xrightarrow{\overline{o}} s_G$, is a *plan* for $\Pi$ if $G \subseteq s_G$. A task $\Pi$ is solvable if a plan exists.

We define the *delete-relaxation* of a task $\Pi$ as the task $\Pi^+$ obtained by setting $\mathsf{del}(o) = \emptyset$, for all $o \in O$ (Hoffmann and Nebel 2001). We say that $\Pi$ is *delete-relaxed solvable* if $\Pi^+$ is solvable. A *relaxed plan for* $\Pi$ is a plan solving $\Pi^+$. The *relaxed facts for* $\Pi$ are a set of instantiated predicates obtained by applying a relaxed plan $\pi^+$ to the initial state, formally $\bigcup_{o \in \pi^+} \mathsf{add}(o)$. Notice that predicates in a set of relaxed facts can naturally be ordered by the first operator in the relaxed plan that included them. We will assume that a set of relaxed facts is represented as a list $l$ that linearize this order. Relaxed plans and their associated lists of ordered relaxed facts can be computed efficiently on the lifted model by using tools like PowerLifted (Corrêa et al. 2020).

Given a PDDL task $\Pi^{PDDL}$, we can compute the corresponding STRIPS task $\Pi$ by *instantiating* the predicates and action schema with the objects in $\Sigma$. Then, $F$ contains a fact for each possible assignment of objects in $\Sigma$ to the arguments of each predicate $P[X] \in \mathcal{P}$, and $O$ contains an operator for each possible assignment of objects in $\Sigma$ to each action schema $a[X] \in \mathcal{A}$. In practice, we do not enumerate all possible assignments of objects in $\Sigma$ to the arguments in facts and action schema. Instead, only those facts and operators are instantiated that are delete-relaxed reachable from the initial state (Helmert 2009). But even doing so, the grounded task can be significantly larger than the original lifted representation, since it grows exponentially in the number of free variables in predicates and action schemas.

We will now briefly summarize the framework for partial grounding we introduced in Gnad et al. (2019), which are based on the grounding algorithm of Fast Downward (Helmert 2006). To ground a planning task, this algorithm performs a fix-point computation, where a queue is initialized with the facts in the initial state and in each iteration one element of the queue is popped and processed. If it is a fact, then operators with preconditions already processed are added to the queue. If it is an operator, all its add effects are added. The algorithm terminates when the queue is empty. We describe the algorithm for STRIPS tasks, but it can be adapted to support other PDDL features (see, e.g., Helmert (2009)).

Algorithm 1 shows our approach. The main difference with respect to the standard algorithm described above is that it can stop before the queue is empty, and operators are instantiated in a particular order. Full grounding terminates only when the queue is empty, ensuring that all delete-relaxed reachable facts and operators are grounded (as delete-relaxation is an over-approximation, this preserves completeness and optimality). Instead, Algorithm 1 can be stopped also by the Stop condition. More importantly, full grounding algorithms extract elements from the queue in an arbitrary order, since order is irrelevant if all operators are grounded. Conversely, Algorithm 1 grounds all

[1] The only limitation is an efficient implementation of the computation of relaxed facts, see below.

**Algorithm 1:** Partial Grounding.

**Input:** A PDDL task $\Pi = (\mathcal{P}, \mathcal{A}, \Sigma^C, \Sigma^O, I, G)$
**Output:** A STRIPS task $\Pi' = (F, O, I, G)$

$q \leftarrow I$ ; $F \leftarrow \emptyset$ ; $O \leftarrow \emptyset$ ;
**while** $\neg q$.empty() $\wedge \neg$*Stop* **do**
  **if** $q$.containsFact() **then**
    $f \leftarrow q$.popFact() ; $F \leftarrow F \cup \{f\}$ ;
    **for** $o \notin O \wedge \mathsf{pre}(o) \subseteq F$ **do**
      $q$.insert($o$) ;
  **else**
    $o \leftarrow q$.popBestOp() ; $O \leftarrow O \cup \{o\}$ ;
    **for** $f \notin F \wedge f \in \mathsf{add}(o)$ **do**
      $q$.insert($f$) ;
**return** $(F, O, I, G)$

facts that have been added to the queue with preference over operators to ensure that effects of grounded operators are part of the grounded task. The popBestOp() function uses a heuristic criterion to determine the order of said operators.

It is important to note that defining the Stop condition is non-trivial. Clearly we should require at least that the goal is included in the set of relaxed facts (i.e., $G \subseteq F$), but this does not ensure plan existence. On the other hand, if the heuristic used by popBestOp() is good (i.e., if relevant operators are consistently chosen) then we know that we can stop "soon" after $G \subseteq F$ holds, since most planning tasks have plans with relatively few operators when compared to the fully grounded set. In this article, we will take a pragmatic approach and define arbitrarily that Stop will hold after $G \subseteq F$ when a predefined number $N$ of operators has been grounded. If search for a plan fails for such $N$, it will be iteratively incremented. In the next section we describe in detail the heuristic used by the popBestOp() function.

## 3 Relaxed Information and Language Models for Grounding

In Gnad et al. (2019) we show two machine learning methods (one based on inductive relational trees and another using classification/regression with relational features) that can be used to define a priority function $f : O \rightarrow [0, 1]$ that estimates whether operators are relevant or not for a given planning task. These models are trained on small, optimally solved planning tasks of a given domain, a set of instances that share the same action schemas, predicates, and constants. Intuitively, they use features from the initial state and goal to predict which action schema instantiations are most relevant. It is important to notice that planning tasks available at training usually have different objects than those encountered during evaluation so that the chosen features cannot refer to specific objects in $\Sigma^O$, and learning has to abstract from specific instance characteristics.

While the approach was successful for many planning domains, the initial state and goal information is limited, since we cannot expect to properly predict the correct instantiation of schemas that act on state features that are not contained in any of these states. To address this shortcoming, we propose to use the list of relaxed facts that describes relevant objects along a complete relaxed plan. In the rest of this section, we will explain how we harvest this information.

Defining machine learning models in terms of lists of relaxed facts presents a number of challenges. Once more, we have the problem of abstracting from the set $\Sigma^O$ of objects that explicitly appear in the list. Moreover, lists will have different sizes, and most machine learning models require a fixed number of features. We propose to address both issues using *natural language models*.

A language model is a probability distribution over sequences of words of a given language like Spanish or English. Given any finite sequence of words $\overline{w}$, a language model assigns a probability $P(\overline{w})$ to it, in terms of the words that appear in $\overline{w}$. Language models learn these probabilities by training on relevant text corpora. However, since languages can be used to express an infinite variety of grammatical sentences, language modeling faces the problem of assigning non-zero probabilities to linguistically valid sequences that may never be encountered during training, and several modeling approaches have been designed to address this issue (applying the Markov assumption, using recurrent neural networks or transformers, etc). Language models are helpful for a variety of problems in computational linguistics, from initial applications in speech recognition (Kuhn and De Mori 1990) to ensure that nonsensical (i.e., low-probability) word sequences are not predicted to wider use in machine translation (Andreas, Vlachos, and Clark 2013) (e.g., scoring candidate translations), information retrieval (Ponte and Bruce 1998), among others.

We will use natural language tools to define *word embeddings* for relaxed facts and operators. Typically, the representation is a real-valued vector that encodes the meaning of the word so that words close to each other in the vector space are expected to be similar in meaning (Jurafsky and Martin 2000). Methods to generate this mapping include neural networks (Mikolov et al. 2013), dimensionality reduction on the word co-occurrence matrix (Levy and Goldberg 2014b), probabilistic models (Globerson et al. 2007), explainable knowledge base methods (Qureshi and Greene 2019), and explicit representation in terms of the context in which words appear (Levy and Goldberg 2014a). Word and phrase embeddings, when used as the underlying input representation, have been shown to boost the performance in NLP tasks such as syntactic parsing (Socher et al. 2013a) and sentiment analysis (Socher et al. 2013b).

We will use fastText, a library developed by Grave et al. (2017) for a wide variety of text classification tasks, to create our word embeddings for both operators and facts. We will now provide a brief outline of the main characteristics of this model. The only requirement to train a fastText model is a corpus of the desired language, namely, a set of plain text sequences. This architecture also has three key hyperparameters: the desired dimension of the embeddings (dim) and the maximum (nmax) and minimum (nmin) of the windows to consider (that will be used both at the character and word level). The input text is processed sequentially, varying the fixed window sizes between nmin and nmax, and a training corpus is created with the characters/words and

their context. Given this training material, a neural network is trained to obtain the probability of co-occurrence between a given character/word, and every other character/word in the vocabulary. The network has a binary input that identifies each of the possible characters/words of the training material through one-hot-encoding, and an output layer with the same dimensionality for the co-ocurrence probabilities, with a hidden layer of size dim connecting both. The word embedding can be obtained from the trained network by simply removing the output layer, as the weights of the hidden layer define the embedding.

Some of fastText's characteristics are particularly useful in our setup. Firstly, when a word is unknown, the character-level model can be used to return a non-null vector, which can help us deal with unknown objects in a planning task that did not appear in our training material. Secondly, the fixed dimensionality of the word embedding will let us define simple machine learning models for classification, independently of the varying length of the list of relaxed facts. Finally, the model is designed for fast querying, which is not usually relevant in many NLP tasks, but is crucial in our setup where the model may be queried millions of times.

## 4 Our Approach

In this section we describe all details needed to create and evaluate the models used to define the behavior of popBestOp() in Algorithm 1. Once a domain $D$ is fixed, the following inputs are required:

**Training Set $\mathbf{Tr}_D$:** A set of "easy" planning tasks used for training.

**Tuning Set $\mathbf{Tu}_D$:** A set of "medium" planning tasks for parameter tuning.

**Evaluation Set $\mathbf{Ev}_D$:** A set of "hard" planning tasks for evaluation. Tasks in $\mathbf{Ev}_D$ are reserved, and they are never available to any of the machine learning steps.

Let $\mathbf{T}_D = \mathbf{Tr}_D \cup \mathbf{Tu}_D \cup \mathbf{Ev}_D$. For each planning task $t$ in $\mathbf{T}_D$ we compute a satisficing plan and a list of relaxed facts. We define the set of *good operators* $\mathbf{GO}_D$ as the set of operators that appears in any plan for a task in $\mathbf{Tr}_D \cup \mathbf{Tu}_D$. We also compute a subset $\mathbf{AO}_D$ of the set of all operators by uniformly sampling a predefined number of operators in $D$, where we ensure that $\mathbf{GO}_D$ and $\mathbf{AO}_D$ are disjoint.

**Language models:** We create two corpora to train language models using fastText. They will not be domain dependent, i.e., we use the same language models for all the domains included in our experiments. One of the corpus collects all plans in $\bigcup_D \mathbf{Tr}_D \cup \mathbf{Tu}_D$, which will be used to automatically define features for operators. The second one collects all lists of relaxed facts in $\bigcup_D \mathbf{Tr}_D \cup \mathbf{Tu}_D$, which will be used to automatically define features for relaxed facts. In all cases, words are defined as strings separated by one or more whitespaces or tabs. Sentences are separated by newlines. Each plan and each list of relaxed facts is considered a sentence, listed in a single line. In all cases, the order of operators and facts is preserved within each line.

To enlarge the size of corpora and ensure sufficient variability of objects, we performed corpora augmentation by adding copies of each sentence where objects are renamed increasing the index in the name of objects[2]. fastText was then run with the following parameters, which are standard:

```
Learning rate: 0.05
Model type: Cont. Bag of Words Model (CBOW)
Epochs: 100
Word Ngrams: 4
Context windows size: 4
Dimensionality: 30
```

The result of this computation is two language models, one that can be used to build representations for facts ($\mathcal{M}_F$), and one for operators ($\mathcal{M}_O$). We can also compute representations for lists of arbitrary size of facts and operators by calculating the average vector with an $l2$ normalization, to make embeddings for sequences comparable to embeddings of single words.

**Regression models:** In principle, it is possible to train a regression model directly using the language models obtained in the previous step. Given each task $t$ in the training set $\mathbf{Tr}_D$, each line in the training matrix is defined by the representation obtained from $\mathcal{M}_F$ ran over the list of relaxed facts associated with $t$ together which the representation obtained from $\mathcal{M}_O$ for an operator $o$ from $\mathbf{GO}_D \cup \mathbf{AO}_D$, labeling the instance as 1 if $o \in \mathbf{GO}_D$ (the operator occurs in a plan), and 0 if $o \in \mathbf{AO}_D$ (not in a plan).

The main problem with the approach described above, which leads to poor performance of the obtained regression model, is that the average size of the lists of relaxed facts for tasks in $\mathbf{Tr}_D$ is orders of magnitude smaller than the corresponding average for lists in $\mathbf{Ev}_D$. Even though their *representation* will have the same size, the averaging algorithm implemented by fastText will produce embeddings that are too far apart, preventing proper generalization.

By design, plans (and their corresponding relaxed fact lists) for tasks in $\mathbf{Tr}_D$ are usually much shorter than plans and lists of relaxed facts for tasks in $\mathbf{Ev}_D$. Moreover, in many cases, they contain "sub-plans" that solve partial goals, and these sub-plans are often interleaved in the solution. A typical example is the Satellite domain where the solution to the easiest task in $\mathbf{Tr}_{\text{Satellite}}$ has 5 actions and 5 relaxed facts, while the easiest plan in $\mathbf{Ev}_{\text{Satellite}}$ has 541 operators and the corresponding list of relaxed facts has size 509. As a result, the distribution of facts is significantly different between training and evaluation.

In light of this fundamental limitation, it is clear that a more complex method to build suitable representations for lists of relaxed facts is required. Imitating the approach used by fastText, we will define running windows over a given list of relaxed facts and use $\mathcal{M}_F$ to build representations for each window. While windows solve the size difference prob-

[2] All domains in our tests are "artificial" and instances are obtained using generators. This ensures that objects are uniformly represented. For example all objects representing stars in Satellite are represented by the word `Star` followed by a natural number $n$. These naming conventions simplify the task of fastText, and domains without this property should be pre-processed accordingly.

lem, it does not solve the *distribution* problem. To address this issue we introduce the notion of *buckets*.

Consider a list of relaxed fact $[f_1, \ldots, f_n]$. First, we divide it into buckets, defined as a set of contiguous facts that use the same predicate, to obtain a list of buckets $[\{f_1, \ldots, f_i\}, \ldots, \{f_j \ldots f_n\}]$. The sampling of windows from these buckets will now be defined by a number of samples (#SAMP), a fixed window size (WSIZE), and step (STEP). We construct a window by randomly picking a fact from WSIZE sequential buckets #SAMP times. The window is then advanced by STEP buckets. We repeat this process for every action in $\mathbf{GO}_D \cup \mathbf{AO}_D$ and combine the representations obtained by applying $\mathcal{M}_F$ to these windows and the operator embeddings produced by $\mathcal{M}_O$ to define $|\mathbf{GO}_D| \times$ #SAMP, $|\mathbf{AO}_D| \times$ #SAMP positive and negative training examples, respectively.

We now seek to train a simple logistic regression model to infer the relevance of an operator with respect to a given window. However, since our training material now contains at least $|\mathbf{Tr}_D| \times |\mathbf{GO}_D \cup \mathbf{AO}_D| \times$ #SAMP samples, it is often infeasible to train the logistic regression model using standard convex solvers, as all data does not fit in memory simultaneously. In order to address this issue we resort to incremental learning using the implementation of stochastic gradient descent (SGD) provided by $scikitLearn$ ($sklearn.linear\_model.SGDClassifier$) (Pedregosa et al. 2011). This estimator implements regularized linear models trained with stochastic gradient descent. The model is updated via the partial fit method based on a learning rate parameter, which allows for mini-batch learning. For our experiments, the option 'log_loss' as the loss function of SGD simply indicates that we train a logistic regression model.

Overall, we use the following standard parameters:

```
wsize: [3,4,5]
step: [3,4,5]
numsamples: 25
loss: "log_loss"
penalty: ["l1", "l2"]
alpha: [0.01, 0.001, 0.0001, 0.00001]
l1_ratio: 0
tol: [0.001, 0.0001]
eta0: 0.0
power_t: 0.5
class_weight: "balanced"
```

$\mathbf{Tu}_D$ is used as evaluation set to perform a grid search over these parameters. Once the best set of parameters is found, the final model is obtained by using $\mathbf{Tr}_D \cup \mathbf{Tu}_D$ as training material. The result of this computation is a logistic regression model $\mathcal{M}_a$ for each action schema $a$ in a domain. The function popBestOp() will then use a priority queue that stores each inserted operator $o$ corresponding to action schema $a$ and assigns its priority by sampling windows from the corresponding set of relaxed facts, constructing the concatenated word embeddings as during training, using $\mathcal{M}_a$ to obtain a relevance score for each (window,operator) pair, and computing the maximum over all windows. We will refer to the full model, including maximization, as $\mathcal{M}_a^{\max}$.

# 5 Datasets

We have generated full datasets for the following planning domains: Agricola, Blocksworld, Depots, Hiking, Satellite, TPP, and Zenotravel. Our choice of domain was somewhat limited by our need of a problem generator with sufficient flexibility to obtain varied problem sets $\mathbf{Tr}_D$, $\mathbf{Tu}_D$, and $\mathbf{Ev}_D$. We selected domains with a relatively small number of action schemas to minimize the number of required models (Agricola being an exception with 22 schemas), and with action schemas with relatively large interfaces for which grounding can be problematic (Blocksworld being now the exception with only up to 3 parameters). In particular, Agricola was chosen as an example of a very demanding and complex domain, while Blocksworld was included as a simple model where full grounding should mostly be feasible.

In our setup, for each domain $D$ we provided at least 150 tasks in $\mathbf{Tr}_D$ which could be solved using Fast Downward (FD) in less than 10 minutes. $\mathbf{Tu}_D$ has at least 50 tasks that required between 10 and 20 minutes of computation to obtain a plan. Finally, we included at least 50 instances in $\mathbf{Ev}_D$ which usually required more than 30 minutes to solve. These numbers are relative, as they are largely hardware dependent, and memory was a stronger constraint than time for some domains.

| | Size $\mathbf{Tr}_D$ | Size $\mathbf{Tu}_D$ | Size $\mathbf{Ev}_D$ |
|---|---|---|---|
| Agricola | 192 | 50 | 108 |
| Blocksworld | 160 | 50 | 63 |
| Depots | 288 | 64 | 64 |
| Hiking | 246 | 130 | 129 |
| Satellite | 243 | 50 | 50 |
| TPP | 220 | 60 | 60 |
| Zenotravel | 294 | 110 | 71 |

$\mathbf{AO}_D$ was defined as a set of $n = 50000$ operators from the set of all operators in $D$. To generate this sample we aim to make the distribution as uniform as possible over action schema (we do not seek to mimic the true distribution of the full set of all operators). In the simplest case, if a domain has $k$ action schema with more than $n/k$ operators in the full set of possible operators, then the sampled subset will contain a random sample of size $n/k$ for each action schema.

The following parameters of the different planning domains we investigated are relevant to estimate the size of the fully grounded set of instantiated actions: total number of action schema (# of A), total number of predicates (# of P), average size of the interface of action schema (Avg. I), and maximal size of the interface of action schema (Max. I).

| | # of A | # of P | Avg. I | Max. I |
|---|---|---|---|---|
| Agricola | 22 | 33 | 4.8 | 8 |
| Blocksworld | 3 | 3 | 2.3 | 3 |
| Depots | 5 | 6 | 3.8 | 4 |
| Hiking | 7 | 9 | 4.6 | 6 |
| Satellite | 5 | 13 | 2.8 | 4 |
| TPP | 4 | 8 | 6.0 | 7 |
| Zenotravel | 5 | 5 | 4.2 | 6 |

All domains and accompanying data, as well as the models uses in our experiments, are available online.[3]

[3] https://gitlab.com/BenjaminOc/grounding_using_small_language_models

## 6 Experimental Results

For the evaluation of our partial grounding approach, we adapted the implementation of the "translator" component of the Fast Downward planning system (FD) (Helmert 2006). The translator parses the given PDDL files and outputs a fully grounded task in finite-domain representation (FDR) (Bäckström and Nebel 1995; Helmert 2009) that corresponds to the PDDL input. Our modifications are minimally invasive, only changing the ordering in which operators are handled and the termination condition, as indicated in Algorithm 1. Therefore, none of the changes affect the correctness of the translator, i. e., the generated grounded planning task will always be a proper FDR task. The changes do not affect the performance too much either, except when using a computationally expensive priority estimation function.

We start by comparing the performance of our models, with respect to a baseline model ($\mathcal{B}$) defined as follows. $\mathcal{B}$ uniformly and randomly assigns a number in the interval $[0, 1]$ to any instantiated operator scheme. Equivalently, $\mathcal{B}$ uniformly samples operators from the full set of available operators without replacement until all the good operators are drawn.

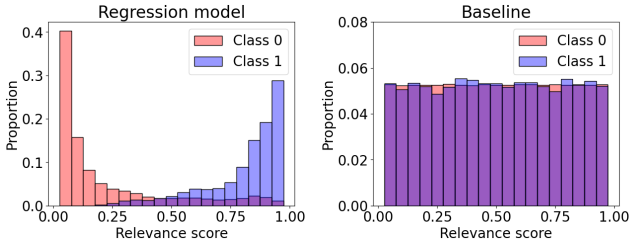


Figure 1: Operator separation comparison in TPP

Figure 1 shows the outcome of the priority function learned by the regression model on TPP, aggregated over all action schemas. The bars indicate the proportion of operators across all tasks in the evaluation set $\mathbf{Ev}_{TPP}$ whose priority falls into a given interval. The class labels 1 and 0 correspond to good ($\mathbf{GO}_D$) and bad operators ($\mathbf{AO}_D$), respectively. The plots nicely illustrate that the priority function seems to work very well in this domain, as the classes are almost perfectly separated. By contrast, and as expected, $\mathcal{B}$ cannot properly separate the classes.

For an in-depth comparison we proceed as follows: for each task $t$ in $\mathbf{Ev}_D$ we consider the set $T$ defined as the union of $\mathbf{AO}_D$ and the operators in the plan $p$ of $t$. We rank all elements in $T$ using first the maximization models $\mathcal{M}_a^{\max}$ for $D$, and then using $\mathcal{B}$. For each of these rankings, we count the number of operators that appear after the last operator of $p$. This is the number of operators that the algorithm can avoid grounding using this ranking. We take the average of this number over all tasks in $t$, and represent it as a percentage w.r.t. the size of $\mathbf{AO}_D$. We call this measure the *average Proportion of Ungrounded Operators* (PUO).

With the aim of investigating how much training data was needed to obtain peek performance for the logistic regression model we train models incrementally over the full set of training instances, in 10% steps, obtaining an $\mathcal{M}_a^{\max}$ for each percentage, and we calculate PUO values for each of these models. Results are shown in Figure 2. To compute the reported p-values we define $X_p, Y_p$ as the PUOs obtained by our model and the baseline model, respectively, for problem $p$, and perform a one-sided paired $t$-test to determine whether the generating distribution of $X_p$ has a larger mean than that of $Y_p$, where we must use a paired test since each tuple $(X_p, Y_p)$ corresponds to the same problem.

| % | Agricola | Blocks | Depots | Hiking | Satellite | TPP | Zenotr |
|---|---|---|---|---|---|---|---|
| 0.1 | 0.9536 | 0.0261 | 0.0965 | 0.8897 | 0.1282 | 0.7871 | 0.4759 |
| 0.5 | 0.9575 | 0.0242 | 0.0957 | 0.8944 | 0.1502 | 0.7836 | 0.4770 |
| 1.0 | 0.9634 | 0.0229 | 0.0965 | 0.8956 | 0.1582 | 0.7878 | 0.4739 |
| Baseline | 0.0070 | 0.0103 | 0.0130 | 0.0130 | 0.0030 | 0.0020 | 0.0020 |
| p-value | 8.34E-127 | 1.24E-04 | 6.70E-17 | 8.34E-127 | 1.05E-21 | 1.64E-87 | 5.39E-95 |

Figure 2: Average Percentage of Ungrounded Operators

| | | | Single Shot | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | novelty | | fastText+SGD | | |
| Domain | | full | FIFO | SQ | RR | SQ | RR | % |
| agricola-large | 25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| agricola-eval | 25 | **5** | 0 | 0 | 0 | 0 | 0 | 0 |
| blocks-large | 25 | 0 | 0 | **25** | 14 | 0 | 21 | 15 |
| blocks-eval | 25 | 22 | 18 | 24 | **25** | 16 | **25** | 23 |
| depots-large | 25 | 1 | 0 | 0 | 0 | 0 | 0 | **1** |
| depots-eval | 25 | **25** | 8 | 2 | 7 | 0 | 11 | 12 |
| hiking-large | 25 | 0 | 0 | 0 | 0 | **7** | 4 | 0 |
| hiking-eval | 25 | 4 | 4 | 4 | 0 | **18** | 13 | 0 |
| satellite-large | 25 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| satellite-eval | 25 | 19 | 21 | 10 | 0 | **21** | 13 | 13 |
| tpp-large | 25 | 9 | 10 | **11** | 5 | 9 | 9 | 9 |
| tpp-eval | 25 | 23 | 22 | **24** | 19 | 11 | 22 | 21 |
| zenotravel-large | 25 | 4 | **25** | **25** | 11 | 4 | 5 | 6 |
| zenotravel-eval | 25 | 22 | **25** | **25** | 17 | 12 | 17 | 16 |
| $\sum$ | 350 | 134 | 133 | **151** | 98 | 98 | 140 | 116 |

Figure 3: Number of instances solved, comparing *full* grounding, a simple *FIFO* ordering that stops when the goal is relaxed reachable, an SQ and RR queue with *novelty*-based action ordering, and our approaches based on fastText+SDG (SQ: single-queue, RR: round-robin, one queue per schema; %: action-plan proportions). All methods do single-shot grounding, runtime limit is 30min.

As it can be seen in Figure 2 results are excellent for all domains, with extremely good $p$-values in all cases (even for Blocksworld, for which we did not expect large improvements). A more surprising conclusion is that the results can be obtained even with little data. This is very encouraging, as it could indicate that our approach can be used in domains where generating training material is difficult.

We also report preliminary testing on using our trained fastText models as "off-the-shelf" language models[4]. We proceeded as follows. Taking TPP as our test case, we trained word embedding models $\mathcal{M}_F$ and $\mathcal{M}_O$ using corpora defined by the training material from all other domains,

[4] As we mentioned in Section 4, the language models we trained were defined using all available training data.

| Domain | | full | Incremental | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | novelty | | fastText+SGD | | |
| | | | FIFO | SQ | RR | SQ | RR | % |
| agricola-large | 25 | 0 | 0 | 0 | 0 | **25** | 2 | 3 |
| agricola-eval | 25 | 5 | 0 | 0 | 0 | **25** | 1 | 6 |
| blocks-large | 25 | 0 | 0 | **25** | 20 | 0 | 23 | 20 |
| blocks-eval | 25 | 24 | 24 | **25** | **25** | 22 | **25** | **25** |
| depots-large | 25 | 1 | 1 | 0 | 0 | **3** | 1 | 1 |
| depots-eval | 25 | **25** | **25** | **25** | **25** | **25** | **25** | **25** |
| hiking-large | 25 | 0 | 0 | 0 | **14** | 7 | 4 | 0 |
| hiking-eval | 25 | 4 | 5 | 4 | **20** | **20** | 16 | 11 |
| satellite-large | 25 | 0 | 0 | **5** | 0 | 0 | 2 | 3 |
| satellite-eval | 25 | 19 | 23 | 23 | 11 | **25** | 21 | 22 |
| tpp-large | 25 | 11 | 10 | 11 | 9 | **17** | 14 | 14 |
| tpp-eval | 25 | 23 | 23 | **25** | 23 | **25** | **25** | **25** |
| zenotravel-large | 25 | 24 | **25** | **25** | 14 | 12 | 11 | 10 |
| zenotravel-eval | 25 | **25** | **25** | **25** | 20 | **25** | 24 | 24 |
| $\sum$ | 350 | 161 | 161 | 193 | 181 | **231** | 194 | 189 |

Figure 4: Same setting as in Figure 3, but with incremental grounding and 90 min runtime limit.

without including a single instance from TPP. We then defined the corresponding $\mathcal{M}^{\max}_{TPP}$ model, using these word embeddings, and only 10% from the available training data from TPP. The obtained PUO value of 0.7805 (with a p-value of 5.6658e-96 w.r.t. the baseline) shows a very small degradation when compared with the corresponding values shown in Figure 2.

PUO values provide a way to evaluate the behavior of the obtained models statistically, while disregarding the possible effects of particular heuristic functions during search and the order in which operators are generated during grounding. But, of course, these factors critically affect which task can or cannot be solved. It is thus important to evaluate how incorporating our models into the grounding process impacts the solver's capabilities to handle large problem instances.

We next take a closer look at the integration of our models into the grounding process of FD, i.e. into its translator component. We extend our previous implementation by replacing the operator priority queue of the grounding step with our trained relevance prediction models. We experiment with three different queue architectures, a single-queue approach (SQ) that keeps operators of all schema in a single sorted queue, a round-robin approach (RR) that has a separate sorted queue for every scheme and alternates between queues, and a proportion-based queue (%). The latter is similar to RR, but weights every scheme queue by the proportion that operators from that scheme appear in the plans in the training set. We use the PowerLifted tool to generate the relaxed facts reached from the initial state (Corrêa et al. 2020). This takes negligible runtime, with a maximum of $1.6s$ on our benchmark set.

Since our models often lead to a too aggressively pruned partial model that does not include a solution, we adopt the incremental-grounding paradigm from our previous work (Gnad et al. 2019). If the search fails to find a solution on the partially grounded model, we go back to grounding and increase its size by instantiating $10.000$ additional operators. We iterate this process until a solution is found or we run out of time. We allow for a total time of 90 minutes and limit each search iteration to 30 minutes. For comparison, we also run single-shot grounding with a total runtime limit of 30 minutes. This variant, like the first iteration of incremental grounding, stops the grounding once the goal becomes relaxed reachable. For all grounding variants, we run the LAMA-first search configuration. We use the large benchmarks from (Gnad et al. 2019) as well as the instances from the evaluation sets $\mathbf{Ev}_D$.[5] For all runs we enforce a memory limit of 4GiB.

| Domain | | full | Single Shot – Minimal Model | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | novelty | | fastText+SDG | | |
| | | | FIFO | SQ | RR | SQ | RR | % |
| agricola-large | 25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| agricola-eval | 25 | **5** | 0 | 0 | 0 | 0 | 0 | 0 |
| blocks-large | 25 | 0 | 0 | **25** | 14 | 0 | 24 | 22 |
| blocks-eval | 25 | 22 | 18 | 24 | **25** | 3 | **25** | 22 |
| depots-large | 25 | **1** | 0 | 0 | 0 | 0 | 0 | 0 |
| depots-eval | 25 | **25** | 8 | 2 | 7 | 6 | 17 | 16 |
| hiking-large | 25 | 0 | 0 | 0 | 0 | **7** | 2 | 0 |
| hiking-eval | 25 | 4 | 4 | 4 | 0 | **19** | 11 | 0 |
| satellite-large | 25 | 0 | 0 | 1 | 0 | 0 | **3** | 2 |
| satellite-eval | 25 | 19 | **21** | 10 | 0 | 16 | 17 | 15 |
| tpp-large | 25 | 9 | 10 | **11** | 5 | 0 | 9 | 6 |
| tpp-eval | 25 | 23 | 22 | **24** | 19 | 0 | 21 | 19 |
| zenotravel-large | 25 | 4 | **25** | **25** | 11 | 9 | 1 | 2 |
| zenotravel-eval | 25 | 22 | **25** | **25** | 17 | 18 | 18 | 18 |
| $\sum$ | 350 | 134 | 133 | **151** | 98 | 78 | 148 | 122 |

Figure 5: Same setting as in Figure 3, but restricting the training data to 10%.

| Domain | | full | Incremental – Minimal Model | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | novelty | | fastText+SDG | | |
| | | | FIFO | SQ | RR | SQ | RR | % |
| agricola-large | 25 | 0 | 0 | 0 | 0 | **25** | 2 | 4 |
| agricola-eval | 25 | 5 | 0 | 0 | 0 | **25** | 1 | 8 |
| blocks-large | 25 | 0 | 0 | **25** | 20 | 0 | **25** | 24 |
| blocks-eval | 25 | 24 | 24 | **25** | **25** | 20 | **25** | **25** |
| depots-large | 25 | **1** | **1** | 0 | 0 | **1** | **1** | **1** |
| depots-eval | 25 | **25** | **25** | **25** | **25** | **25** | **25** | **25** |
| hiking-large | 25 | 0 | 0 | 0 | **14** | 7 | 4 | 4 |
| hiking-eval | 25 | 4 | 5 | 4 | **20** | **20** | 16 | 18 |
| satellite-large | 25 | 0 | 0 | **5** | 0 | 1 | 3 | 3 |
| satellite-eval | 25 | 19 | **23** | **23** | 11 | 22 | **23** | 22 |
| tpp-large | 25 | 11 | 10 | 11 | 9 | 15 | **17** | **17** |
| tpp-eval | 25 | 23 | 23 | **25** | 23 | 24 | **25** | **25** |
| zenotravel-large | 25 | 24 | **25** | **25** | 14 | 15 | 6 | 8 |
| zenotravel-eval | 25 | **25** | **25** | **25** | 20 | 24 | 23 | 21 |
| $\sum$ | 350 | 161 | 161 | 193 | 181 | **224** | 196 | 205 |

Figure 6: Same setting as in Figure 4, but restricting the training data to 10%.

In Figures 3 and 4 we show the number of solved instances (coverage) on our fastText + SGD models using

[5] As PowerLifted does not currently support conditional effects, we could not evaluate our approach on the Caldera domain.

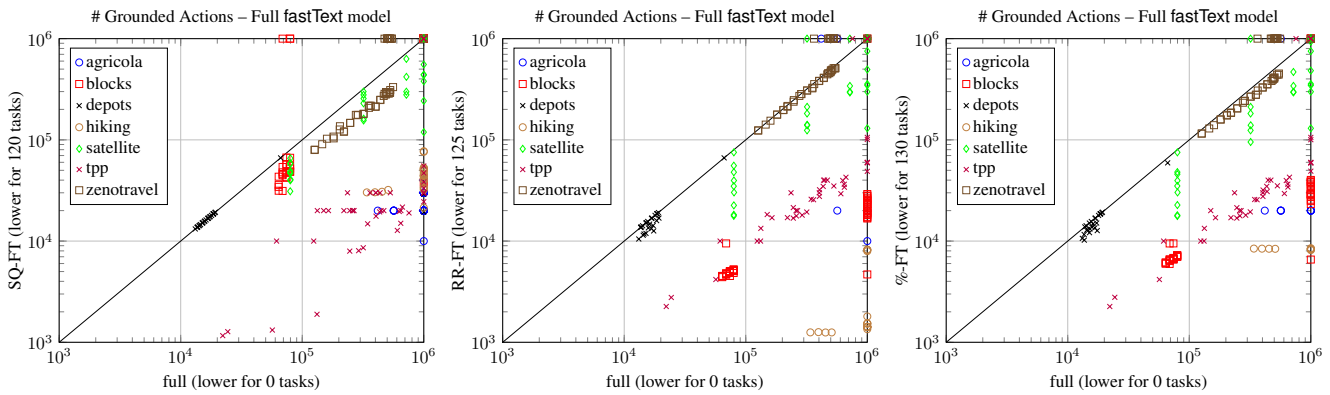


Figure 7: Per-instance plots comparing the number of grounded operators of our fastText + SDG approach to the fully grounded model. Only instances solved by at least one approach are shown. All methods get a runtime limit of 90 minutes, our approaches run incremental grounding.

all training data. Figure 3 shows the single-shot grounding, comparing our approach to the full grounding (full) and a simple FIFO queue that stops once the goal is relaxed reachable; both do not use any operator relevance analysis. We also include two configurations using a novelty-based priority function from our prior work, with a SQ and RR queue. As already mentioned, it looks like our models prune the tasks too aggressively, so quite often no solution can be found on the partially grounded model in the single-shot setting. However, moving to the incremental grounding setup clearly changes those results, since in this setting the models learned based on fastText representations significantly outperform the baseline algorithms. This is particularly clear for the SQ queue, which has especially good performance in Agricola and TPP. It also seems like there is a substantial overhead when using fastText compared to, for example, the simple novelty metric. Our language-model based configurations benefit significantly from the higher overall runtime in the incremental setting (90 vs. 30 minutes), while this advantage is less pronounced for the other approaches. Hence, for domains where grounding is a bottleneck, allowing for some more time to evaluate our models does pay off and allows to solve challenging instances.

In Figures 5 and 6 we show a similar evaluation, but with the fastText + SGD models trained on only 10% of the data. Quite surprisingly, the effect of reducing the amount of training data is not negative overall. While the SQ queue seems to suffer most, with exceptions in the evaluation instances of Depots and Zenotravel, the other queues actually have a higher total coverage with these models. This is great news, as it indicates that for certain domains and algorithm configurations, only little training data is required to train high-quality models.

Finally, we investigate the size of the partially grounded models, comparing it to the full grounding when using the entire training set. Scatter plots that compare the number of ground operators per instance are shown in Figure 7. For every instance in our benchmark set, we show the size of the fully grounded model in the $x$-axis, and the one obtained using our models with the three different queues on the $y$-axis. Hence, the farther points are below the diagonal, the smaller the partially grounded model is compared to the full grounding. In every plot, we only include instances that were solved by at least one of the two methods. An immediate observation is that all our models achieve a great reduction in the size of the grounding. The specific reduction factors vary across domains, but the behavior is very consistent across queue types. With up to more than two orders of magnitude, we achieve the highest reduction in Agricola, Hiking, and TPP, agreeing with our statistical tests in Figure 2. Except for the SQ queue, we also get a significant reduction in Blocksworld and a good reduction in Satellite. The reduction is only minor in Zenotravel and Depots, which explains relatively weak coverage performance of our approaches in these domains compared to the baselines. We also evaluated our models obtained from only 10% of training data (not shown due to space restrictions), which remarkably show only very minor differences in comparison to the full models.

Overall, our evaluation shows that small-language models based on fastText are indeed capable of learning good priority functions to estimate the relevance of instantiated operators during grounding. The models generalize well to instances of significantly larger size, even if limiting the training data to only 10%.

## 7 Conclusion

In this paper we build on previous work that introduced partial grounding for automated planning (Gnad et al. 2019) based on relevance prediction using classical machine learning models. The main limitation of these models was their reliance on solely the initial and goal states to assess operator relevance. We propose the use language models (word embeddings) to leverage the intermediate information available in the list of relaxed facts that can be efficiently computed from the initial state of a planning problem.

## References


Andreas, J.; Vlachos, A.; and Clark, S. 2013. Semantic parsing as machine translation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, volume 2.

Bäckström, C.; and Nebel, B. 1995. Complexity Results for $SAS^+$ Planning. *Computational Intelligence*, 11(4): 625–655.

Corrêa, A. B.; Francès, G.; Pommerening, F.; and Helmert, M. 2021. Delete-Relaxation Heuristics for Lifted Classical Planning. In Goldman, R. P.; Biundo, S.; and Katz, M., eds., *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)*, 94–102. AAAI Press.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Francès, G. 2020. Lifted Successor Generation using Query Optimization Techniques. In Beck, J. C.; Karpas, E.; and Sohrabi, S., eds., *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)*, 80–89. AAAI Press.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Francès, G. 2022. The FF Heuristic for Lifted Classical Planning. In Honavar, V.; and Spaan, M., eds., *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI 2022)*, 9716–9723. AAAI Press.

Corrêa, A. B.; and Seipp, J. 2022. Best-First Width Search for Lifted Classical Planning. In Thiébaux, S.; and Yeoh, W., eds., *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)*, 11–15. AAAI Press.

Fikes, R. E.; and Nilsson, N. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2: 189–208.

Fišer, D. 2020. Lifted Fact-Alternating Mutex Groups and Pruned Grounding of Classical Planning Problems. In Conitzer, V.; and Sha, F., eds., *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*, 9835–9842. AAAI Press.

Globerson, A.; Chechik, G.; Pereira, F.; and Tishby, N. 2007. Euclidean Embedding of Co-occurrence Data. *Journal of Machine Learning Research*, 8(76): 2265–2295.

Gnad, D.; Torralba, A.; Dominguez, M.; Areces, C.; and Bustos, F. 2019. Learning How to Ground a Plan – Partial Grounding in Classical Planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 7602–7609.

Grave, E.; Mikolov, T.; Joulin, A.; and Bojanowski, P. 2017. Bag of Tricks for Efficient Text Classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017n*, 427–431. Spain.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26: 191–246.

Helmert, M. 2009. Concise Finite-Domain Representations for PDDL Planning Tasks. *Artificial Intelligence*, 173: 503–535.

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 14: 253–302.

Jurafsky, D.; and Martin, J. 2000. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*. Prentice Hall.

Kuhn, R.; and De Mori, R. 1990. A cache-based natural language model for speech recognition. *IEEE transactions on pattern analysis and machine intelligence*, 12(6): 570–583.

Lauer, P.; Torralba, Á.; Fišer, D.; Höller, D.; Wichlacz, J.; and Hoffmann, J. 2021. Polynomial-Time in PDDL Input Size: Making the Delete Relaxation Feasible for Lifted Planning. In Zhou, Z.-H., ed., *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI 2021)*, 4119–4126. IJCAI.

Levy, O.; and Goldberg, Y. 2014a. Linguistic Regularities in Sparse and Explicit Word Representations. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, 171–180. Ann Arbor, Michigan: Association for Computational Linguistics.

Levy, O.; and Goldberg, Y. 2014b. Neural Word Embedding as Implicit Matrix Factorization. In Ghahramani, Z.; Welling, M.; Cortes, C.; Lawrence, N.; and Weinberger, K., eds., *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL – The Planning Domain Definition Language – Version 1.2. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, Yale University.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.; and Dean, J. 2013. Distributed Representations of Words and Phrases and their Compositionality. In Burges, C.; Bottou, L.; Welling, M.; Ghahramani, Z.; and Weinberger, K., eds., *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

Pednault, E. P. 1989. ADL: Exploring the Middle Ground Between STRIPS and the Situation Calculus. In Brachman, R.; Levesque, H. J.; and Reiter, R., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the 1st International Conference (KR-89)*, 324–331. Toronto, ON: Morgan Kaufmann.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830.

Ponte, J.; and Bruce, C. 1998. A language modeling approach to information retrieval. In *Proceedings of the 21st ACM SIGIR Conference*, 275–281. ACM.

Qureshi, M.; and Greene, D. 2019. EVE: explainable vector based embedding technique using Wikipedia. *Journal of Intelligent Information Systems*, 53: 137–165.

Richter, S.; Westphal, M.; and Helmert, M. 2011. LAMA 2008 and 2011 (planner abstract). In *IPC 2011 planner abstracts*, 50–54.

Ridder, B.; and Fox, M. 2014. Heuristic Evaluation Based on Lifted Relaxed Planning Graphs. In Chien, S.; Fern, A.; Ruml, W.; and Do, M., eds., *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 244–252. AAAI Press.

Sievers, S.; Röger, G.; Wehrle, M.; and Katz, M. 2019. Theoretical Foundations for Structural Symmetries of Lifted PDDL Tasks. In Lipovetzky, N.; Onaindia, E.; and Smith, D. E., eds., *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)*, 446–454. AAAI Press.

Socher, R.; Bauer, J.; Manning, C. D.; and Ng, A. Y. 2013a. Parsing with Compositional Vector Grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 455–465. Sofia, Bulgaria: Association for Computational Linguistics.

Socher, R.; Perelygin, A.; Wu, J.; Chuang, J.; Manning, C. D.; Ng, A.; and Potts, C. 2013b. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 1631–1642. Seattle, Washington, USA: Association for Computational Linguistics.

Wichlacz, J.; Höller, D.; and Hoffmann, J. 2022. Landmark Heuristics for Lifted Classical Planning. In De Raedt, L., ed., *Proceedings of the 31st International Joint Conference on Artificial Intelligence (IJCAI 2022)*, 4665–4671. IJCAI.